Human-Help in Automated Planning Under Uncertainty

Ignasi Andrés Franch

Tese apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Doutor em Ciências

Programa: Doutorado em Ciências da Computação
Orientadora: Profa. Dra. Leliane Nunes de Barros

Projeto de Pesquisa financiado pela CAPES

São Paulo, setembro de 2018

Human-Help in Automated Planning Under Uncertainty

Esta versao da dissertaçao/tese contém as correções e alterações sugeridas pela Comissão Julgadora durante a defesa da versão original do trabalho, realizada em 21/09/2018. Uma cópia da versão original está disponível no Instituto de Matemática e Estatística da Universidade de São Paulo.

Comissão Julgadora:

- Profa. Dra. Leliane Nunes de Barros (presidenta) - IME-USP
- Prof. Dr. Denis Deratani Mauá - IME-USP
- Profa. Dra. Karina V. Delgado - EACH-USP
- Prof. Dr. Valdinei Freire - EACH-USP
- Prof. Dr. Carlos Ribeiro - ITA

Agradecimentos

A la meua familia, els meus pares Maria Isabel i Jose Ignacio i les meues germanes Maria i Fatima, per tot el suport i comprensió.

Als meus cosins i oncles: Ferran, Mireia, Maria, Joan, Enric, Rosa Maria, Paco, Vicent i Mari Carmen.

A tots els meus amics: Eugeni, Joan, Ciscar, Modesto, Tonio, Sergio, Anton, Jose Luis i les respectives families, per fer més lleuger i suportable el viatge, i per organitzar uns sopars inoblidables.

À minha orientadora, professora Leliane, pela paciência, os ensinamentos, as contribuições e a direção de este trabalho.

Aos professores Denis e Karina, pelas contribuições feitas para este trabalho.

Aos amigos dos laboratorios do IME muito obrigado pelos momentos e dicas.

À galera do Quinta & Games pelos momentos de diversão, descontraimento e novas ideias.

À NTU e à Innovative, muito obrigado por terem me dado as boas oportunidades que tive com voces.

A tudo o mundo que tive prazer de conhecer e que torceu por mim.

E por último, áquela sem quem nada disso seria possível, à minha mulher Paula.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001

Resumo

Planejamento é a subárea de Inteligência Artificial que estuda o processo de selecionar ações que levam um agente, por exemplo um robô, de um estado inicial a um estado meta. Em muitos cenários realistas, qualquer escolha de ações pode levar o robô para um estado que é um beco-sem-saída, isto é, um estado a partir do qual a meta não pode ser alcançada. Nestes casos, o robô pode, *pró-ativamente*, pedir ajuda humana para alcançar a meta, uma abordagem chamada *autonomia simbiótica*. Neste trabalho, propomos duas abordagens diferentes para tratar este problema: (I) *planejamento contingente*, em que o estado inicial é parcialmente observável, configurando um *estado de crença*, e existe não-determinismo nos resultados das ações; e (II) *planejamento probabilístico*, em que o estado inicial é totalmente observável e as ações tem efeitos probabilísticos. Em ambas abordagens a ajuda humana é considerada um recurso escasso e deve ser usada somente quando estritamente necessária.

No *planejamento contingente*, o problema é encontrar uma política (mapeamento entre estados de crença e ações) com: (i) garantia de alcançar a meta (política forte); (ii) garantia de *eventualmente* alcançar a meta (política *forte-cíclica*), ou (iii) sem garantia de alcançar a meta (política *fraca*). Neste cenário, uma das contribuições deste trabalho é propor sistemas de planejamento contingente que considerem **ajuda humana para transformar políticas fracas em políticas fortes (cíclicas)**. Para isso, incluímos ajuda humana de dois tipos: (i) ações que modificam estados do mundo e/ou estados de crença; e (ii) observações que modificam estados de crenças.

Em *planejamento probabilístico*, o problema é encontrar uma política (mapeamento entre estados do mundo e ações) que pode ser de dois tipos: *política própria*, na qual o agente tem probabilidade 1 de alcançar a meta; ou *política imprópria*, caso exista um beco-sem-saída inevitável. O objetivo do agente é, em geral, encontrar uma política que minimize o custo esperado acumulado das ações enquanto maximize a probabilidade de alcançar a meta. Neste cenário, este trabalho propõe sistemas de planejamento probabilístico que considerem **ajuda humana para transformar políticas impróprias em políticas próprias**, porém considerando dois novos critérios: *minimizar a probabilidade de usar ações do humano* e *minimizar o número esperado de ações do humano*. Mostramos ainda que políticas ótimas sob esses novos critérios podem ser computadas de maneira eficiente considerando que ações humanas possuem um custo alto ou penalizando o agente ao pedir ajuda humana.

Soluções propostas em ambos cenários, planejamento contingente e planejamento probabilístico com ajuda humana, foram empiricamente avaliadas sobre um conjunto de problemas de planejamento com becos-sem-saida. Os resultados mostram que: (i) todas as políticas geradas (fortes (cíclicas) ou próprias) incluem ajuda humana somente quando necessária; e (ii) foram encontradas políticas para problemas de planejamento contingente com até 10^{15000} estados de crença e para problemas de planejamento probabilístico com até $3 * 10^{18}$ estados do mundo.

Abstract

Planning is the sub-area of artificial intelligence that studies the process of selecting actions to lead an agent, e.g. a robot, to a goal state. In many realistic scenarios, any plan of actions can lead the robot into a dead-end state, that is, a state from which the goal cannot be reached. In such cases, the robot can, pro-actively, resort to human help in order to reach the goal, an approach called *symbiotic autonomy*. In this work, we propose two different strategies to tackle this problem: (I) *contingent planning*, where the initial state is partially observable, configuring a *belief state*, and there is non-determinism in the outcomes of the robot actions; and (II) *probabilistic planning*, where the initial state may be partially or totally observable and the actions have probabilistic outcomes. In both scenarios, the human help is considered a scarce resource that should be used only when necessary.

In **contingent planning**, the problem is to find a policy (a mapping from states to actions) that: (i) guarantees the agent always reaches the goal (*strong* policy); (ii) guarantees that the agent *eventually* reaches the goal (*strong cyclic* policy), or (iii) does not guarantee achieving the goal (*weak* policy). In this scenario, one of this work contributions is to propose contingent planning systems that consider **human help to transform weak policies into strong (cyclic) policies**. To do so, two types of human help are included: (i) human actions that modify states and/or belief states; and (ii) human observations that modify belief states.

In **probabilistic planning**, the problem is to find a policy (a mapping from world states to actions) that can be one of these two types: a *proper policy*, where the agent has a probability 1 of reaching the goal; or an *improper policy*, in the case of unavoidable dead-ends. In general, the goal of the agent is to find a policy that minimizes the expected accumulated cost of the actions while maximizes the probability of reaching the goal. In this scenario, this work proposes probabilistic planners that consider **human help to transform improper policies into proper policies** however, considering two new (alternative) criteria: either *to minimize the probability of using human actions* or *to minimize the expected number of human actions*. Furthermore, we show that optimal policies under these criteria can be efficiently computed either by increasing human action costs or given a penalty when a human help is used.

Solutions proposed in both scenarios, contingent planning and probabilistic planning with human help, were evaluated over a collection of planning problems with dead-ends. The results show that: (i) all generated policies (strong (cyclic) or proper) include human help only when necessary; and (ii) we were able to find policies for contingent planning problems with up to 10^{15000} belief states and for probabilistic planning problems with more than $3 * 10^{18}$ physical states.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Robots are already part of our lives, being used in factories to automate industrial operations, helping people to perform tasks; and in hostile environments like deep sea mining, space operations or search and rescue operations (USAR). In factories robots are already able to perform their tasks autonomously, for example, putting pieces of cars together or painting them, but to increase their autonomy, they must be able to interact with humans or workers. In the soon future, service robots will not only perform tasks that humans view as cumbersome, but they will be also part of our everyday life to perform tasks like driving, delivering products and providing assistance to elderly. In human hostile environments, like space operations, teams of humans and robots should be able to collaborate with each other to accomplish their missions. In sum, robots should execute all their tasks in the presence or even under the guidance of humans.

In particular, as population ages worldwide, there will be need for health care services for the aged population. The shortage of professionals in that area, e.g. nurses or caregivers for elderly is a great opportunity for service robots. However, even if robots are widely accepted to perform repetitive tasks, they are still regarded with suspicion when it comes to interact with elder people (European Comission, 2012). Nonetheless, this seems to be changing in countries like Japan (Andrew Tarantola, 2017), and US where *aging in place*, that is, to live in their own home instead of retiring to a facility, is preferred. These robots will have to help elderly people interacting with them (Frank Tobe, 2012). As an example, consider an *elderly assistant robot* (Georgia Institute of Technology, 2012), able to help elder people by performing daily chores like cleaning the kitchen, doing laundry, taking out the trash, reminding the elder to take its medication, or help the elder to move as a walking frame. In all those cases, it is possible that some human help is required, like to ask the human to open a door, or open a drug flask.

Another example of a service robot for household tasks is the *Home Assistant Robot AR* (Johou Systems Kougaku (JSK) Laboratory, 2012), a robot able to move in a domestic environment and perform chores (Yamazaki et al., 2012, 2010). This robot is able to perform the chores of the house according to some pre-scheduled assignment. However, this plan may depend on some conditions, and an unexpected event might prevent the robot to complete their tasks, e.g. a child or a pet near the robot that can be harmed. In this case the robot should wait until the room is free, or allow the robot to ask its human to either help it to liberate the room, or assign the robot for another task.

Collaboration between humans and robots (or software agents) is an interesting field of study. They involve robots acting along with humans and planning *together* to reach a goal. This raises the question about who should make the decisions, the human controller or the robot? The *naive* answer to this question would, on one hand, have the human controlling the robot by **teleoperation**, which would minimize the robot autonomy, and would need a dedicated human supervisor for each task; on the other hand, having the robot deciding all its actions, an approach known as **full robotic autonomy**, requires a robot completely capable to solve all problems in all situations. A better answer to this question would consist in having each actor deciding when it is best to relinquish its autonomy.

Several approaches for human robot collaboration have been proposed. For example, **symbiotic autonomy** (Veloso et al., 2015) models a robot that proactively and autonomously asks for human help when needed; **adjustable autonomy** (Scerri et al., 2002) allows for the robot and the human to divide the responsability for the success of the task. An interesting type of **adjustable autonomy** for our work is the **mixed-initiative** approach (Côté et al., 2012; Crandall and Goodrich, 2001; Mouaddib et al., 2009, 2010), where the initiative is shared between the robot and a human supervisor. In the case of a domestic robot, it should be left performing its task alone until finding a situation that it is unable to solve, and only in such situation ask for human help. However, a **mixed-initiative** approach would rely on a human supervisor to help the robot, without the robot indicating its need for help.

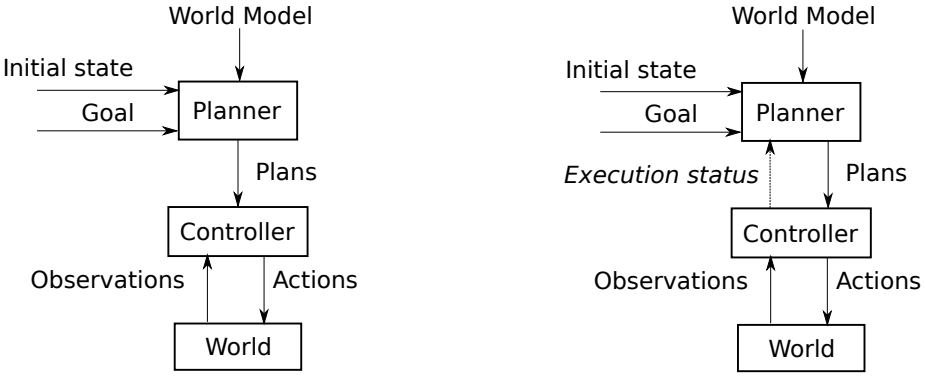


Figure 1.1: *Conceptual model for planning problems (extracted from Ghallab et al. (2004)). (left) offline planning; (right) online planning.*

Tasks involving planning *agents* have to select actions based on anticipating their effects in order to generate a plan of actions that achieves a certain goal. The robot is fed with a description of a model of the environment and their capabilities in a formal action language, for example PDDL (Younes and Littman, 2004). A plan can involve several robot actions that must be translated into robot actuations (or sensors reading). Fig. 1.1 (left) depicts this basic architecture. In this work, we are only concerned in **task planning**.

Planning the best actions to be performed can be done before the robot is deployed in the environment, in what is called *offline* planning (Fig. 1.1 left); or the robot can interleave planning and execution of actions, that is called *online* planning (Fig. 1.1 right). On the downside, computing a full *offline* plan of actions may require more time and resources, whereas the robot may compute the best action to perform very fast in an *online* fashion.

To effectively solve complex robotic tasks, the robot should plan considering the uncertainty in the environment. That is, the robot should plan under uncertainty w.r.t. the effects of the actions (non-deterministic or probabilistic effects), as well as in the missing information about the world state (partial observability). In scenarios as complex as those, it is often realistic to consider the possibility of the robots failing to accomplish a task, due to different circumstances. For example, a robot can end up in a state from which it is impossible to reach its mission goal. This state, called *dead-end* state, while not inevitable it might occur with some possibility or probability. These states can be caused by several reasons, for example the description of the world is incomplete; the robot gets into a situation that was not predicted; or because the robot is not capable to perform a task. For example, the elderly assistant robot may not be able to open a drug bottle or open the bathrooms door.

Traditional solutions to solve planning problems intend to maximize the autonomy of the robot, but it does not specify a behavior if a dead-end is ever encountered. In this work, we are interested in computing plans, including human actions when necessary, to overcome *dead-end* states, i.e., possible failures of a robotic task.

It is then possible to devise a robot that not only plans, but also finds where it is possible to

encounter a failure, and asks a (possibly) human supervisor to correct this problem only when this problem occurs. Contrary to the *mixed-initiative* approaches, the responsibility lies entirely with the robot. Fig. 1.2 shows the difference between our approach and *mixed-initiative*. In the *mixed-initiative* approach, depicted in the top of Fig. 1.2, the human or the robot make their own plans, meaning that once the control is with either one of them (the human or the robot), the other one cannot interfere. In our approach, depicted in the bottom of Fig. 1.2, however, the robot makes all plans, including the human plans. Hence, the human is merely seen as an external executor or helper reducing the need to modeling the workload of the human and the need for *situational awareness*. Notice the gradient color indicating where the robot is more autonomous.

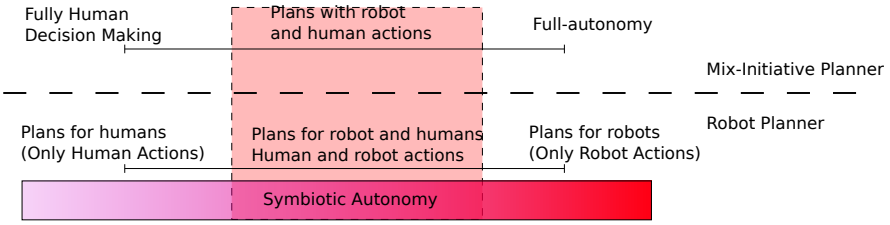


Figure 1.2: *Differences between a mixed-initiative planner approach (top), and a different approach where the robot plans for both the robot and the human (bottom).*

1.1 Contingent Planning

Contingent planning (Geffner and Bonet, 2013; Hoffmann and Brafman, 2005) models a robot in a partially observable environment with non-deterministic actions. When planning with partial observation, a solution can be found searching in a space of *belief states*, where a belief state b is represented by a set of possible world states $s \in b$. Contingent planning allows the planner to explicitly reason about multiple outcomes and possible scenarios and where the probability of the outcomes is not known a priori. Contingent planning problems can have three types of solutions: a *weak* solution, with no guarantees to achieve the goal; a *strong* solution, which guarantees to achieve the goal and; a *strong cyclic* solution that eventually guarantees to achieve the goal, despite the cycles. We say that π is a strong (cyclic) plan for a contingent problem P, iff every execution of π is applicable and finishes in a belief state b where the goal holds $\forall s \in b$. We say that π is a weak solution of a contingent problem P, iff there is at least one execution of π that finishes in a dead end, i.e. a state from which the agent cannot achieve the goal. A contingent planning problem can be solved by a search in an AND/OR graph of belief states where the nodes are of two types: AND nodes and OR nodes. An OR node represents a belief state; and an AND node represents all the effects of an action or observation.

1.2 Probabilistic Planning

Goal-oriented Markov Decision Processes (GMDPs) are the standard framework for building mission guided planning robots under uncertainty or with stochastic transitions (Bertsekas and Tsitsiklis, 1991; Kolobov et al., 2012; Teichteil-Königsbuch, 2012; Trevizan et al., 2017). The typical strategy in this framework is to maximize the probability of reaching the goal while minimizing the expected accumulated cost. GMDP can have two types of solutions: a *proper* solution, that reaches the goal with probability 1; or an *improper* solution, where the probability is less than 1 of reaching the goal. A GMDP can be solved by either performing a *synchronous dynamic programming algorithm* like the *Value Iteration* algorithm that updates the value function of *all* states in each iteration (Bertsekas and Tsitsiklis, 1991), or by *asynchronous* algorithms that compute policies only for the reachable states from the initial state (Bonet and Geffner, 2003).

Although this approach generates policies that are robust and cost-effective, it assumes the robot will simply abort the mission whenever a dead-end state is encountered. An arguably better approach when facing a dead-end is to reach out for human help.

1.3 Objectives

The main objective of this thesis is to solve planning problems with unavoidable dead-ends, proactivelly including in the solution human actions and observations in order to have a policy that guarantees to reach the goal.

Moreover, we assume that human help actions are a scarce resource to be used only if necessary. Thus, another objective of this work is to find optimal (or sub-optimal) policies that only include human help when strictly necessary. Finally, a third objective is to propose efficient methods to solve planning problems with human help.

1.4 Thesis Proposal

1.4.1 Human Help in Contingent Planning

In this work we present a novel way for planning with partial observation and non-deterministic actions, where a strong (cyclic) solution may not exist, due to the presence of dead-ends, i.e. problems for which there is only weak policies. In this setting, we propose an approach that allows an agent to plan considering human help of two forms: human observations and human actions (changing in the environment), with the objective of generating a strong (cyclic) solution when, otherwise, only a weak policy exists.

Because in most of the problems it might not be possible to model which human help actions will be available, the agent must automatically infer them from the description of the problem. Thus, human actions can be applied in any state and modify a single proposition from the state description. Yet, human observations are obtained from the robot's own set of observations, relaxing their preconditions. Accessing a human observation allows the robot to disambiguate belief states eliminating its uncertainty about the world. Since we assume that human help actions are to be used only if necessary, these actions and observations have a higher cost than the robot actions an therefore are used by the robot only when necessary.

The proposed contingent approach with human help, can deal with different types of dead-end belief states, including pure dead-end states (e.g. a broken robot) and dead-ends that arise due to uncertainty about the environment that cannot be solved using only the robot's own observations or actions. Moreover, we propose methods to consider only the relevant set of human actions in order to promote efficiency.

1.4.2 Human Help in Probabilistic Planning

We also consider the problem of goal-oriented probabilistic planning with human help actions (no human observations are required for these problems since we assume an agent with full observalibity). We propose a generalization of Goal-Oriented Markov Decision Processes (GMDPs), augmented with human help actions, called (GMDP-HH), where the robot has a distinguished set of human help actions that can be applied in any state modifying a single fluent, which are also automaticly extracted from the problem specification.

There are no optimality criteria defined for such problems, hence, we first define a criteria to minimize the probability of using human-help. While appealing, this criterion is difficult to attain, as the corresponding Bellman equation has multiple non-optimal fixed points, and heuristics to estimate probability are usually inefficient (Trevizan et al., 2017). Thus, we consider an alternative class of decision problems, called **GMDPs with a Penalty on Human Help (GMDP-PHH)**, where a finite penalty is incurred only the first time a human help is used. An optimal policy minimizes

the expected accumulated cost (which includes the penalty for using a human help for the first time).

However, as we will show, minimizing the probability of using human action encourages the robot to maximize the number of human actions whenever a human action is used, since this decreases the cost of robot actions and incurs no additional cost and maximizes the probability of reaching the goal. To alleviate such problem, we propose another criterion, to minimize the expected number of actions, breaking ties by the expected accumulated cost of robot actions. We show that optimal policies under this criterion can be obtained by optimizing for a proxy criterion that assigns a large uniform cost over human actions and then minimizes the expected accumulated cost (of robot and human actions).

1.5 Contributions

The main contributions of this thesis are:

- Formalization of contingent planning including human observations and actions automatically extracted from the problem description (Andrés et al., 2017; Franch, 2017);
- A characterization of dead-end belief states for contingent planning (Andrés et al., 2017; Franch, 2017);
- Use of the well-known classical planning structures, *Domain Transition Graph* and the *Causal Graph*, of a determinization of a contingent or probabilistic planning problems, to compute a minimal set of human actions that guarantees the existence of strong (cyclic) or proper solutions (Andrés and de Barros, 2016);
- Development of a contingent planning system with human help, called HH-CP planner that translates a contingent planning problem to a FOND problem described in an epistemic language which can be solved by an efficient *off-the-shelf* FOND planner (Andrés et al., 2017);
- Development of a contingent planning system with human help, called COMP2BT+HH, that extracts the smallest set of human actions that although incomplete, is more efficient than HH-CP;
- An empirical comparative analysis of HH-CP and COMP2BT+HH systems over a set of benchmark domains (Andrés et al., 2017);
- Formalization of probabilistic planning problems including human actions, automatically extracted from the problem description (Andrés et al., 2018b);
- Formalization of probabilistic planning problems given a set of human actions; and
- Definition of optimization criteria for probabilistic planning with human help to minimize the use of human help and an empirical analysis of their performance (Andrés et al., 2018a).

1.6 Related Work

Most of the work on human-robot interaction is based on probabilistic planning with partial observation over the environment and the human competences and availability (Armstrong-Crews and Veloso, 2007; Cai et al., 2009; Doshi et al., 2008; Jaulmes et al., 2005; Karami et al., 2009; Kearns and Singh, 2002; Rosenthal et al., 2011; Schmidt-Rohr et al., 2008). A simple solution is to augment a POMDP (*Partially Observable Markov Decision Process*) with a set of human observations (and/or actions) with negative reward and the objective is to find a policy that maximizes the expected reward over a given horizon. Usually, the human set of observations and actions are different from the robot set of observations and actions. For example,

the CoBot (Rosenthal et al., 2010; Veloso et al., 2015) asks humans for help when the robot has too much uncertainty about its location or when it is uncertain of which action to take (Rosenthal et al., 2010).

Another approach to solve these problems is to consider *mixed-initiative* based solutions (Côté et al., 2012; Crandall and Goodrich, 2001; Mouaddib et al., 2009, 2010). In general these decisions are taken during the execution (online planning). In this context, giving the initiative means not only to let the robot or the human execute the actions, but also that the responsibility to compute the plan lies in both, and that the robot must synchronize its actions with those of the human when executing such plan. These approaches have also interesting properties, as they guarantee not only the minimal probability to use these special actions, but also minimize its expected number. But these approaches focus in executing the solutions *online*, instead of calculating the solution *offline*. For example, consider a situation where a robot acting in the environment computes a plan, in an *offline* fashion. Then it starts executing it and if it encounters a problem (*dead-end*), it will transfer the initiative to the human supervisor, while *discarding* the plan it has computed before. When the human supervisor gets the initiative, it can control the robot by teleoperation, following a plan he has calculated, instead of using the robot's calculated plan.

When dealing with *mixed-initiative* based solutions, it is important to remark the notion of *situation awareness*. The *situation awareness* is the measure of how well the human perceives the environment within a volume of space and time (Endsley, 1988), and impacts heavily the performance of the teleoperation (Yanco and Drury, 2004), hence, a robot should make clear why it has failed, to help the human overcome the problems the robot encountered.

1.7 Organization of this Thesis

We divided this thesis in Part I (Contingent Planning with Human Help) and Part II (Probabilistic Planning with Human Help). We start by giving a brief review on classical planning (Section 2).

In Part I, we focus on using human help in planning with non-deterministic effects, that is FOND and POND planning problems. In Chapter 3 we start by describing FOND problems and how to solve these problems. We continue by presenting POND planning problems, i.e., conformant and contingent planning problems, and how to solve them by translating these problems to a FOND problem. We also present a new categorization of dead-ends, taking into account how the uncertainty in the initial situation may cause them, and show examples of them. Next, in Chapter 4 we introduce human help, describing how can be used to obtain strong solutions for every one of these classes of dead-ends. We also introduce formally the domain transition and causal graph, that shows the dependencies between the literals of the problem and show how to compile information from these graphs to obtain a relevant set of literals that can be used as human help actions. Then, in Chapter 5, we show experiments that validate empirically our proposal, and we also evaluate the scalability, comparing our approach to the state-of-the-art in contingent planning. And in Chapter 6 we review the related work, like Symbiotic Autonomy, and other solutions for contingent planning with dead-ends.

In Part II we shift our focus to probabilistic planning problems. In Chapter 7 we explain formally probabilistic problems, describing current approaches to deal with dead-ends in these problems. In Chapter 8 we will introduce a new class of problems, where the robot has a distinguished set of human help actions that can be applied in any state. And we propose three optimization criteria. Then, in Chapter 9, we evaluate the scalability of our proposal and investigate the characteristics of policies obtained with these different criteria in augmented versions of standard planning domains (Doors, Tire World and Navigation). Finally, in Chapter 10 we review the related work, like Symbiotic Autonomy, and other solutions based on Mixed Initiative Planning.

Chapter 2

A Brief Review of Automated Planning

In this chapter we make a brief review on the classical planning theory that in this thesis will serve for two important purposes: (1) to present the algorithms for classical planning that will be later used as heuristics to solve contingent and probabilistic planning problems (Section 2.1.5); and (2) to introduce the basic concepts of planning under uncertainty as an extension of classical planning (Section 2.2).

2.1 Classical Planning

Artificial intelligence is the branch of computer science that studies the intelligent behavior of agents, that is, the ability of an agent to make rational decisions to optimize a certain performance measure. In this context, automated planning is an important research area of artificial intelligence that studies the ability of an agent to choose a sequence of actions that leads to a desired goal state. The *General Problem Solver* (Newell et al., 1959) is considered the first planning system and is one of the oldest programs of Artificial Intelligence .

Among the sub-areas of automated planning, *classical planning* is the most restrictive setting, which makes two restrictive assumptions: actions have deterministic effects and the environment is fully observable. A classical planning problem consists in a set of states, an initial state, a set of goal states and a set of actions that the agent can perform in the environment.

The solution of a classical planning problem is a plan, i.e. a sequence of actions that, when executed in the initial state, leads to a goal state (Ghallab et al., 2004).

Example 1 (Dock-Worker Robot)**.** *In this planning domain, truck robots must move containers among piles in different locations, also using a set of cranes to load/unload the trucks. The initial situation describes the localizations of the containers, trucks and cranes. The actions are: (1)* ***move*** *the truck from one location to other; (2)* ***take*** *a container with a crane; (3)* ***put*** *down a container on a pile, (4)* ***load*** *the truck; (5)* ***unload*** *a truck. The goal is a given configuration of containers among the piles.*

The dynamics of the environment can be seen as a graph, called *transition state graph*, where the set of vertexes correspond to states, and the edges represent the actions. Figure 2.1 shows the transition state graph for a small instance of the Dock-Worker Robot domain, where s_0 is the initial state, and s_5 is the goal state. For example, applying action *move1* in the state s_0 moves the truck from *location2* to *location1*, resulting in state s_2. Thus, solving a classical planning problem can be seen as a problem of finding the shortest path between two vertex s_0 (initial state) and s_5 (goal state), that is, the minimal sequence of actions (assuming unitary cost of actions, otherwise the solution would be the sequence with minimal accumulated cost), between s_0 and s_5, which in the Figure 2.1 is the sequence $\{move1, take, load, move2\}$.

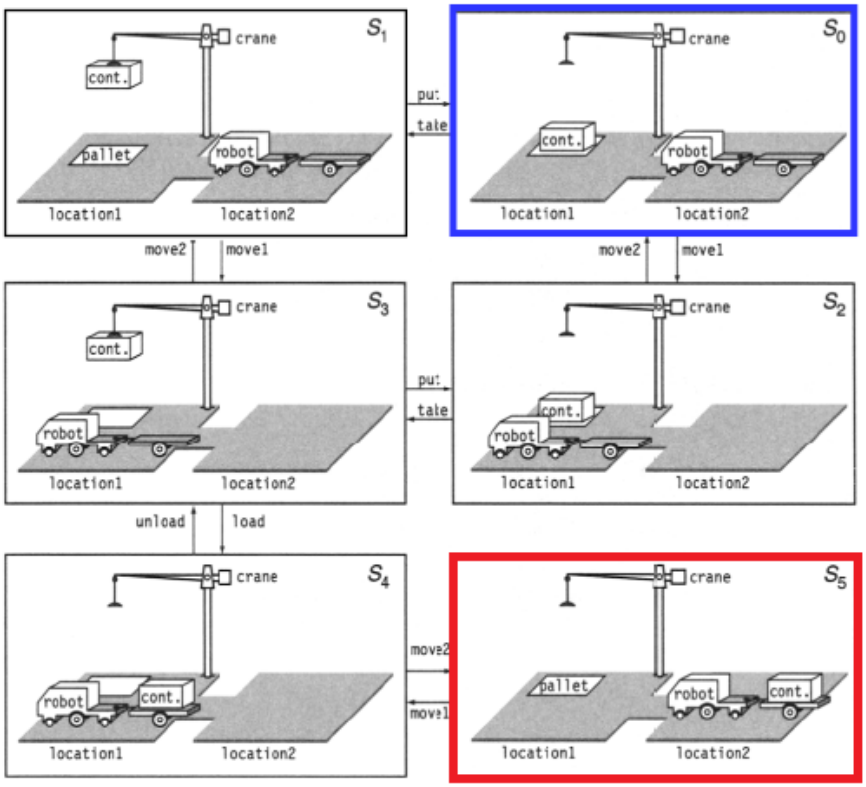


Figure 2.1: *Example of an instance of the Dock-Worker Robot planning domain; with 1 truck, 1 crane, 1 pile, 1 container and 2 locations (Extracted from Ghallab et al. (2004)).*

2.1.1 Classical Planning Model

The model underlying a classical planning problem is a state transition graph with edges labeled with the actions and nodes labeled by states. This model describes the dynamics of the environment, as well as the goal of the agent, the initial state and the action cost function. Formally, this model is given by a tuple $\mathcal{M} = \langle S, s_0, S_G, \mathcal{A}, \mathcal{T}, \mathcal{C} \rangle$ (Geffner and Bonet, 2013):

Definition 1 (Classical Planning Model)**.** *A classical planning model is a tuple* $\mathcal{M} = \langle S, s_0, S_G, \mathcal{A}, \mathcal{T}, \mathcal{C} \rangle$*:*

- *a finite set of states* S*,*
- *a known initial state* $s_0 \in S$*,*
- *a set of goal states* $S_G \subseteq S$*,*
- *a set of actions* $\mathcal{A}$ *such that* $\mathcal{A}(s) \subseteq \mathcal{A}$ *represent the set of actions applicable in a state* $s \in S$*,*
- *a deterministic state transition function* $\mathcal{T} : S \times \mathcal{A} \to S$*, representing the state resulting from executing an action* $a \in A$ *in a state* $s \in S$*.*
- *a cost function* $\mathcal{C} : \mathcal{A} \to \mathbb{R}^+$*;* $\mathcal{C}(a)$ *defining the cost of applying action* a *for every action* $a \in \mathcal{A}(s)$*.*

◇

A plan π consists of a sequence of actions $a_0, a_1, ..., a_{n-1}$ that, when applied in an initial state s_0, generates a sequence of states $s_0, s_1, ..., s_n$, with s_n being a goal state, i.e., $s_n \in S_G$. In classical planning it is not necessary to perform observations during the execution of the plan, because the agent knows exactly which is the state after executing an action in a state. The cost of a plan of

size n is the accumulated cost of its pairs $(\pi(s_i))$, i.e., $cost(\pi) = \sum_{i=0}^{n-1} \mathcal{C}(\pi(s_i))$. An optimal plan π^* has the minimum cost among all plans that start in s_0 and end in $s \in S_G$. A classical automated planning system takes as input $\mathcal{M} = \langle S, s_0, S_G, \mathcal{A}, \mathcal{T}, \mathcal{C} \rangle$, and produces a plan π^*.

2.1.2 The STRIPS planning language

Two main classes of planning languages are used to encode a classical planning model: *Boolean* and *multi-valued* classes, according to the type of variables used to encode states. In the first class, the state variables are Booleans, i.e only two values are accepted **True** or **False**. In the second class of languages, variables can be multi-valued assuming different values from a finite domain. In this work we focus mainly on boolean variables.

The oldest planning language is STRIPS (Fikes and Nilsson, 1972), originally based on a first order predicate logic and is considered to be the first planning language. A set-theoretic representation version of STRIPS is the one used by most of the planning systems (even beyond classical planning). Boolean variables are also called propositions (or atoms).

Definition 2 (Set-theoretic STRIPS Planning Problem)**.** *A classical planning problem expressed in* STRIPS *is a tuple of the form* $\mathcal{M}_{STRIPS} = \langle F, I, Op, G \rangle$*, in which:*

- F *is a set of atoms or propositions of the problem,*
- $I \subseteq F$ *is a set of propositions defining the initial situation,*
- Op *is a set of operators specifying the set of actions* $\mathcal{A}$ *(which along with* F *define a planning domain), and*
- G *is a set of propositions* $f \in F$ *that must be satisfied in a goal state.*

◇

A state s in STRIPS is given by the set of propositions that are true in s, assuming that every proposition not in the set s is false. This is called the *Closed World Assumption* (CWA). Every action $a \in Op$, is represented by the lists: $prec(a)$, $add(a)$ e $del(a)$ of propositions. The list $prec(a)$ contains propositions that must hold in the state s before executing action a; the list $add(a)$ contains propositions that must be true in the state after executing a; and the list $del(a)$ contains propositions that must be false after executing a. We assume w.l.o.g. that actions are **well formed**, that is, $add(a) \cap del(a) = \emptyset$ and $del(a) \subseteq prec(a)$.

A proposition $f \in F$ appearing in the effect of an action is a *fluent*, i.e., f is a proposition whose truth value can be changed by the agent. On the other hand, a proposition that does not appear in any effect is called an *invariant* because its value never changes.

A STRIPS problem encoded in the form of a tuple $\mathcal{M}_{STRIPS} = \langle F, I, A, G \rangle$ defines a classical planning model $\mathcal{M} = \langle S, s_0, S_G, \mathcal{A}, \mathcal{T}, \mathcal{C} \rangle$, where:

- the set of states S are all the possible combinations of truth values over the set F, where each state $s \in S$ is represented by the subset of propositions of F that are true in S.
- the initial state s_0 is I,
- the set S_G of goal states contain all states $s \in S$ such that $G \subseteq s$,
- actions $a \in \mathcal{A}(s)$ correspond to operators in Op such that $prec(a) \subseteq s$,
- the state transition function $\mathcal{T}(a, s) = (s \setminus del(a)) \cup add(a)$, i.e. the state resulting of applying action a in s, is the state s removing the propositions in $del(a)$ and with the addition of the propositions in $add(a)$.
- the cost function $\mathcal{C}$ returns always 1.

Since states in S are formed by collections of propositions from the set F, the number of possible states is up to $2^{|F|}$ which is an upper bound, since it also includes states that are not reachable, i.e. states the agent will never visit applying all possible actions (some of them are inconsistent indeed).

A problem in STRIPS describes actions by means of schemas over generic propositions using predicates names, e.g., in the Dock-Worker Robot the predicate $free$ and variables like ?$crane$ representing objects in the environment (Geffner and Bonet, 2013). This schema operators must be grounded, that is, the variables are replaced by the object names. We call each grounded predicate a proposition.

2.1.3 Example of a planning problem in STRIPS

This example, for compactness, is described in the original STRIPS language based on first-order logic. Let us consider the Dock Worker Robots problem depicted in Figure 2.1 where there is the following objects: 1 crane (cr), 1 container (co), 1 truck (t), 1 pile (p), a pallet (pa) on top of the empty pile (pa) and 2 different locations ($loc1$ and $loc2$). The predicates are defined in Table 2.1. The actions are: move, take, put, load and unload. Thus, the STRIPS description of this problem given in terms of $\langle F, I, Op, G\rangle$ is as follows:

- $F = \{at(cr, loc1), at(cr, loc2), at(co, t),\ at(co, p),\ at(p, loc1),\ at(p, loc2),\ at(t, loc1),\ at(t, loc2),\ adj(loc1, loc2),\ adj(loc2, loc1),\ occupied(loc1),\ holds(co, cr),\ occupied(loc2),\ free(p),\ loaded(t, c),\ top(p, pa),\ on(pa, co),\ unloaded(t),\ free(cr)\ \}$,
- $I = \{at(cr, loc1),\ free(cr),\ unloaded(t),\ at(p, loc1),\ on(pa, co),\ at(t, loc2),\ adj(loc1, loc2),\ adj(loc2, loc1), top(p, co)\}$,
- $Op = \{\ move(t, loc1, loc2),\ take(co, cr, p),\ putDown(co, cr, p),\ load(co, cr, t),\ unload(co, cr, t)\ \}$
- $G = \{at(co, t),\ at(t, loc2)\}$.

The STRIPS operators are described in terms of the tuples $\langle prec(a), add(a), del(a)\rangle$ as shown in Table 2.2. For example, the action $move$ allows the truck to move between adjacent locations ($loc1$ and $loc2$), only if the destination is not occupied:

```
Action move(loc1, loc2):
  Precond: at(t,loc1), adj(loc1,loc2), ¬occupied(loc2)
  add: at(t,loc2), occupied(loc2)
  del: at(t,loc1), occupied(loc1)
```

To obtain a set-theoretic representation for this example we have to instantiate (to ground) all operators which will result in 64 propositions and 288 actions. Note that before planning there is a consistency check needed, because some of this fluents or actions are inconsistent.

The complete set of actions of Table 2.2 together with F is a complete specification of the Dock Worker Robot planning domain. Notice that the number of propositions of the problem depends on the number of objects considered. Every fluent expresses the position or state of the objects on the domain, e.g., $at(co, t)$ expresses that the container is at the truck. And each one of these fluents is considered to be true if it represents a fact of the world. For example, in the initial situation $at(co, cr)$ is true and means that the crane holds the container. The set of states S, is formed by the different propositions and represents the position in which the agent is located.

A possible solution for this problem could be:

```
{ move(t, loc1, loc2),
take(co1,pa,cr,p,loc1),
load(co, cr, t, loc1),
move(t, loc2, loc1)}
```

Predicates	Description
at(?object, ?location)	The ?object is at ?location
holds(?container, ?crane)	The ?container is hold by the ?crane
adj(?location1, ?location2)	Two locations ?location1 and ?location2 are adjacent
occupied(?location1)	There is no object at ?location
free(?crane)	The ?crane is not holding any object
top(?pile, ?object)	The ?object is at the top of the ?pile
on(?object1, ?object2)	?object1 is in top of ?object2
loaded(?truck, ?container)	The ?truck has ?container loaded
unloaded(?truck)	?truck has no load

Table 2.1: STRIPS *schemas representing predicates described over generic propositions for the Dock Worker Robots.*

Action	Parameters	Preconditions	Add List	Del List
move	truck - t, locations - loc1, loc2	at(t,loc1), adj(loc1,loc2), ¬occupied(loc2)	at(t,loc2), occupied(loc2)	at(t,loc1), occupied(loc1)
take	object - co1,co2, crane - cr, pile - p, location -loc1	at(cr,loc1), at(p,loc1), free(cr),top(p,co1) ,on(co1,co2)	holds(co1,cr),top(p,co2)	free(cr),at(co1,p), top(p,co1),on(co1,co2)
put	object - co1,co2, crane - cr, pile - p, location - loc1	at(cr,loc1), at(p,loc1), holds(co1,cr), top(p,co2)	free(cr), at(co1,p), top(p,co1),on(co1,co2)	holds(co1,cr), top(p,co2)
load	object - co, crane - cr, truck - t, location - loc1	at(cr,loc1), at(t,loc1), holds(co, cr), unloaded(t)	free(cr),loaded(t,co)	holds(co, cr), unloaded(t)
unload	object - co, crane - cr, truck - t, location - loc1	at(cr,loc1), at(t,loc1), loaded(t,co),free(cr)	holds(co,cr), unloaded(t)	free(cr), loaded(t,co)

Table 2.2: STRIPS *operators (actions) described in terms of the tuples of the Dock Worker Robots.*

2.1.4 Multi-Valued Planning Language: SAS$^+$

The multi-valued planning language SAS$^+$ (Bäckström and Nebel, 1995; Jonsson and Bäckström, 1998) defines a set of variables and their possible values. Actions in this languages are operators that change the partial assignment over the variables representing the next state:

Definition 3 (Multi-valued planning tasks)**.** *A multi-valued planning task* SAS$^+$ *is a tuple* $\mathcal{M}_{\text{SAS}^+} = \langle \mathcal{V}, I, G, \mathcal{O} \rangle$*, where:*

- $\mathcal{V}$ *is a finite set of state variables, each with its own associated domain* D_v*;*
- I *is a state defining the initial situation;*
- G *is a partial assignment of variables, over a subset of variables* $\mathcal{V}' \subseteq \mathcal{V}$ *that must hold in the goal states; and*
- $\mathcal{O}$ *is a set of operators over* $\mathcal{V}$*. An operator* a *is a tuple* $a = \langle prec(a), eff(a) \rangle$*, where* $prec(a)$*, called the preconditions of the action, is a partial assignment of variables; and* $eff(a)$*, called effects, is a set of pairs* $\langle v, d \rangle$*, where* $v \in \mathcal{V}$ *is a variable called affected variable, and* $d \in D_v$ *is called the new value for* v*.*

◇

A *partial variable assignment* over $\mathcal{V}$ is a function $\int$ on a subset of variables of $\mathcal{V}$, such that $\int(v) \in D_v$ if $s(v)$ is defined. A state, is a partial variable assignment defined for all variables in $\mathcal{V}$.

To illustrate this language, consider the Dock Worker Robots problem depicted in Figure 2.1. The SAS$^+$ description of the action *move* expressed in the SAS$^+$ language is:

```
Action move(loc1, loc2):
  prec(a): at(t) = loc1, adj(loc1,loc2) = true, occupied(loc2) = false
  eff(a): ⟨at(t), loc2⟩ ,
  ⟨occupied(loc2), false⟩
```

where $at(t)$, $adj(loc1, loc2)$ and $occupied(loc2)$ are multi-valued variables, representing, respectively, the location of the truck t, if two locations $loc1, loc2$ are adjacent and if a location is occupied, and the domain D_v for each variable is:

- $D_{at(t)} = \{l_1, l_2\}$,
- $D_{adj(loc1,loc2)} = \{true, false\}$, and
- $D_{occupied(loc2)} = \{true, false\}$.

Notice that in SAS$^+$ language the preconditions are partial variable assignments that must be tested to check if the precondition holds, and the effects of the actions are assignments of values to variables.

A multi-valued planning problem problem encoded in the form of a tuple $\mathcal{M}_{\text{SAS}^+} = \langle \mathcal{V}, I, G, \mathcal{O} \rangle$ defines a classical planning model $\mathcal{M} = \langle S, s_0, S_G, \mathcal{A}, \mathcal{T} \rangle$, where:

- the set of states S are all the possible combinations of partial variable assignments, where each state $s \in S$ is represented by a partial variable assignment $\int$ defined for all variables $v \in \mathcal{V}$;
- the initial state s_0 is defined by the assignments of I;
- the set S_G of goal states is a partial variable assignment g over V, such that for all states $s \in S_G$, $g \in s$;
- actions $\mathcal{A}(s) \in \mathcal{O}$ correspond to operators in $\mathcal{O}$ such that $prec(a) \subseteq s$;
- the state transition function $\mathcal{T}(a, s)$, i.e. the state resulting of applying action a in s, is the state s' where:
  - $s'(v) = d$ if there exist an effect $eff(a) = \langle v, d \rangle$, or
  - $s'(v) = s(v)$ otherwise.

2.1.5 Solving Classical Planning Problems in STRIPS

The state-of-the-art solution in classical planning is based on heuristic search. Heuristics estimate the distance of the successor state s' to the goal state, by solving a relaxed version of the problem, where the initial state is s' and the heuristic value is the size (or accumulated cost) of the optimal solution for the relaxed planning problem (Bonet and Geffner, 2001; Helmert, 2006; Hoffmann and Nebel, 2001).

There are several efficient heuristic search algorithms to solve classical planning, among them, the most known are *Fast-Forward* (FF) (Hoffmann and Nebel, 2001) and *Fast-Downward* (FD) (Helmert, 2006) (both winners in the International Planning Competition (IPC)[1]). The FF planner uses as heuristic the size of the optimal plan for a relaxed version of the problem solved by the GraphPlan planner (Blum and Furst, 1997). The relaxation consists in ignoring the $del(a)$ list of all actions $a \in A$.

GraphPlan

GraphPlan is a planning system that works in two phases. In the first phase, *GraphPlan* computes the planning graph $\mathcal{G}$, and in the second phase it extracts a partial ordered plan, if such a plan exists.

The planning graph is a layered graph that alternates two different kinds of layers: *facts* layer and *actions* layer. A facts layer i consists of all literals that can be reached after i steps; and

[1] IPC: http://icaps-conference.org/index.php/Main/Competitions

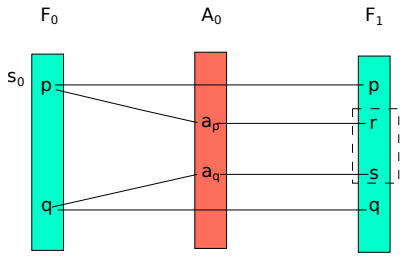


Figure 2.2: *Planning Graph example where F_0 is the first facts layer, and A_0 is the first actions layer; F_0 includes all literals describing the initial state; A_0 includes all actions applicable in F_0; F_1 includes all literals added by the actions in A_0; the dotted lines in F_1 represent the goal literals. The solution to this simple example is the plan $\langle a_p, a_q \rangle$ or $\langle a_q, a_p \rangle$.*

the action layer consists of all actions that can be applied in the previous facts layer. Figure 2.2 shows an example of planning graph. The algorithm starts by including in the first facts layer F_0, all the literals describing the initial state. Then, the action layer A_0, includes all actions whose preconditions are present in the previous facts layer. An edge going from a literal l_1 in the facts layer F_0 to an action a in the actions layer A_0, means that l_1 appears in the preconditions of a. Then, a new facts layer F_1 is created, containing all the literals appearing in the previous facts layer F_0 plus the literals appearing in the effects of all actions in A_0. Each time a new layer is generated, the *mutexes* between literals and actions are marked.

Two actions are marked to be in *mutex*, if there is no possible ordering between them, such that one action does not interfere with the other, that is:

Inconsistent effects: the effect of one action negates the precondition of another.

Interference: an action deletes the precondition of another action.

Competing needs: two actions have literals of its preconditions marked as mutex.

As an example consider the Mutex M_1 in Figure 2.3 where two actions a_p and a_q, with preconditions p and q, respectively, such that $\neg p \in del(a_q)$, and $\neg q \in del(a_p)$, that is, its effects are inconsistent. And there exists two different types of *mutexes* for the literals:

Inconsistent support: two literals are marked to be in *mutex* if one is the negation of the other, for example consider the mutex M_2 in Figure 2.3 between literals p and its complement $\neg p$ appearing in the same facts layer.

Competing effects: two literals are marked to be in *mutex* if they were added to the graph by two actions marked to be in mutex, for example consider the mutex M_3 in Figure 2.3 between literals $\neg p$ and $\neg q$, added by actions a_q and a_p, respectively, which are in mutex.

GraphPlan expands the graph until:

- all the literals of the goal G are present in a facts layer without being marked in a mutex; or
- two consecutive facts layers are equal, meaning that the algorithm has reached a fixed-point and there is no solution.

Once the planning graph has been created, the solution plan is extracted by performing a regressive depth-first search from the last facts layer. An action is selected for each literal of the goal establishing the list of preconditions of all selected actions as sub-goals. This process repeats until it reaches the initial facts layer F_0. In the solution, actions selected from the same layer do not have a total order relation. If a literal is selected such that it is in mutex with other literals

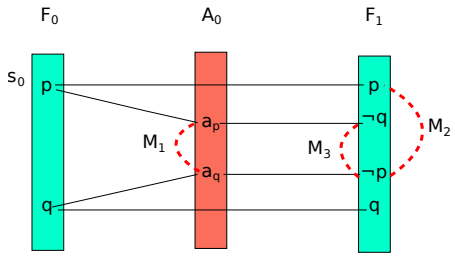


Figure 2.3: *Planning Graph example for a problem with fluents p and q, and actions a_p and a_q, with different types of mutex between actions and literals.*

already selected, the algorithm discards this literal, and backtracks to the previous layer, selecting other actions. If there is no possible action to select without being in mutex with another action, Graphplan returns a failure or expands the graph if it has not.

Figure 2.2 shows an example of the planning graph computed by the Graphplan algorithm, where the initial state is formed by two fluents p, q and there exist two actions applicable a_p and a_q, which add r and s, respectively.

FF-Planner

The FF-Planner (Hoffmann and Nebel, 2001) is a planning system that performs a heuristic search in the state space. The heuristic, called h_{FF}, is the size of the optimal solution, using GraphPlan, of a relaxed version of the problem. The problem's relaxation is to ignore the negative effects of all actions. As output it returns the size of the optimal computed plan, or ∞ if no plan exists.

Formally, given a planning problem $P = \langle F, I, A, G\rangle$ h_{FF} computes the optimal solution of the relaxed planning problem $P^+ = \langle F, s_i, A^+, G\rangle$, where each action a in A^+ has no *del* list, and s_i is the new initial state considered at each search node. This relaxed version, solved by the *GraphPlan* planner, is guaranteed to have no mutexes (Hoffmann and Nebel, 2001), because there will be no conflict between literals and between actions, hence the creation of this graph is faster, making FF-Planner a very efficient algorithm for classical planning.

2.1.6 Solving Classical Planning Problems in SAS$^+$

Solving classical planning problems expressed in SAS$^+$ can also be based on heuristic search that applying actions described in a multi-valued planning language. In these problems, applying an action on a state s will generate a successor state s' where each variable will have its value modified according to the effects of the actions.

In this section we will describe a process that comprises the computation of two graphs: the *domain transition graph* and the *causal graph* (Helmert, 2006). The first graph is the *Domain Transition Graph* for each multi-valued variable. This graph encodes the circumstances that make a variable change its value, more specifically from which values $d \in D_v$ of a variable v there exists a transition to other variables. The second is the *Causal Graph*, that encodes the dependencies between the different multi-valued variables. This graph can be cyclic, however since this graph was designed to be used as an heuristic, it is possible to obtain a relaxed acyclic version. This acyclic version consists in a relaxation of the original planning task where the preconditions of an operator are ignored. Then, we will describe the FD planner that uses both graphs as heuristic, to compute an estimate of the distance to the goal, by counting how many changes must be performed to the values of the variables, until reaching the goal.

Domain Transition Graph

The Domain Transition Graph (DTG) of a variable v shows the relation between the different values $d \in D_v$ of a variable (Helmert, 2006; Jonsson and Bäckström, 1998). A DTG represents under which conditions these values change.

Definition 4 (Domain Transition Graph). *Let $\Pi = \langle \mathcal{V}, s_0, s_G, \mathcal{O} \rangle$ be a multi-valued planning task, and $v \in \mathcal{V}$ be a variable of Π. The Domain Transition Graph (DTG) of a variable v, denoted by DTG(v) is a labeled directed graph (V, E) with the set of nodes $V = D_v$, and for all pairs of values $d, d' \in D_v$ there exists an edge between d and d' if:*

- *There exists an effect $\langle cond, v, d' \rangle$, where $pre \cup cond$ contains a condition $v = d$. In this case, the edge is labeled as $pre \cup cond \setminus \{v = d\}$.*
- *There exists an effect $\langle cond, v, d' \rangle$, where $pre \cup cond$ does not contains a condition $v = d$ for any $d \in D_v$. In this case, the edge is labeled as $pre \cup cond$.*

◇

The labels of the edges are called the *conditions*. Informally, this graph represents that in the DTG(v), there is an edge between two values d and d', if there is a possible action that change the value of v from d to d'. As an example of DTG for the variables *pcont* and *ptruck* of the Dock Worker Robots encoding, respectively, the position of the container and the truck, consider Figure 2.4 left top and left bottom.

In every multi-valued problem there is a correspondence between the DTGs of all the variables and the state space of the problem. The execution of the plan can be seen as simultaneous node traversal in the DTG. In each step, only a node of each DTG(v) for all $v \in \mathcal{V}$ represents the value of the variable. This node is called the *active node*, and applying an operator means to change which is the active node for each graph.

Causal Graph

Definition 5 (Causal Graph). *Let $\mathcal{M}_{\text{SAS}^+} = \langle \mathcal{V}, s_0, s_G, \mathcal{O} \rangle$ be a Multi-valued planning task. The Causal Graph of $\mathcal{M}_{\text{SAS}^+}$, denoted by CG$(\mathcal{M}_{\text{SAS}^+})$ is a directed graph (V, E) with the set of nodes $V = \mathcal{V}$, and such that there exists an edge between two variables v and v', s.t. $v \neq v'$ if:*

- *There exists an edge in the DTG of v' with a condition on v; or*
- *v and v' appear in some effect $\langle cond, v, d \rangle$ of the same operator.*

◇

In the first case, there is an edge because v' depends on v for changing a value, and in the second case because they are *co-occurring* effects. Informally, there is an edge (v, v') between two variables if changes in one variable v' depend on the values of other variable v, or both variables change value at the same time.

As an example of CG of the Dock Worker Robots variables, consider Figure 2.4 (right). It is easy to see the dependencies between the variables. in this example, it can be seen that the position of the container (*pcont*) depends on the position of the truck (*ptruck*), if it is loaded, or the position of the crane (*pcrane*).

Acyclic Causal Graphs

The causal graph can contain cycles, meaning that all variables are dependent on the values of other values. To solve a multi-valued planning task efficiently with the use of the DTG and the CG it is necessary for the graph to be acyclic. However, in Helmert (2006) both graphs are meant to serve as an heuristic, so it is possible to transform the original cyclic Causal Graph, by pruning

some edges that results in ignoring some preconditions. But, the more preconditions are relaxed, the worse the heuristic estimates the distance to the goal.

The idea of the relaxation is to create an *ordering* $\prec$ on the variables $v \prec v'$, such that the *higher level variable* v' retains its outgoing edges whenever there is a cycle between v' and the *lower level variable* v, i.e. we retain in the Graph only those edges v, v' such that $v \prec v'$ The greedy algorithm that transforms a cyclic CG to an acyclic one is shown below:

- Compute the DTG and the CG, and assign a weight n for each edge, where n is the number of operators that induce this edge.
- For the strongly connected elements of the graph, select the vertex v with a the minimal accumulated weight of incoming edges, and set $v \prec v'$ for all the other vertex v' of the graph.
- Remove the vertex v and its edges from consideration.
- repeat iteratively until one vertex is left.

Once a vertex v has been selected, we prune the DTG by removing from the labels all conditions on variables v' that include v. These correspond to the *relaxed* preconditions.

FD-Planner

The FD-Planner (Helmert, 2004, 2006) is also an heuristic planning system that solves a classical planning problem by performing a search in the state space. The heuristic, in this case is not the size of the relaxed version of a problem, but the accumulated costs of changing the values of the fluents appearing in the goal. To compute this in a fast way, this planner uses the two graphs explained before: the *domain transition graph* and the *causal graph*.

As an example of how these graphs are used to compute the heuristic, consider the Dock Worker Robots. In Figure 2.4 are depicted the domain transition graph, and causal graph for the variable *pcont*, *ptruck* and *pcrane* that encodes the position of the container, the truck and the crane, respectively. To illustrate the causal graph heuristic, consider the initial and goal states, as depicted in Figure 2.1, and the graphs in Figure 2.4. Then, let us imagine the active node in each domain transition graph at the initial state, that is, the node *pile* for the position of the container, and the node *loc2* for the position of the truck. The container is at the top of the pile, and it must be put in the truck to be moved to location 2. To do so, it must be *taken* (action take) by the crane, changing the active node from *pile* to *crane*. The causal graph shows a dependency between the position of the container and the position of the truck. Thus, moving the truck must be taken into account to accomplish the preconditions as indicated by the labels on the edge of the domain transition graph. Once the truck is in location 1, the container can be loaded on the truck, and the truck can move to location 2. The cost of this plan is 4, that is the number of edges we have traversed in the different domain transition graphs.

2.2 Non Classical Planning

Classical planning assumes the environment only evolves in a totally predictable way, i.e., it assumes deterministic effects and complete observation of the world (Ghallab et al., 2004). However, some problems may have uncertainty in the effects, that is, an action may have different outcomes with or without preference over them. Also the planning agent might have partial observability, or not observability at all, of the world's initial state. Table 2.3 shows the different types of non classical planning problems that are characterized according with the assumptions they make about the world observability and the uncertainty on the action effects. There exist two main types of uncertainty in the effects of the actions: (1) *knightian* uncertainty (FOND); and (2) *probabilistic* uncertainty. In both cases there are approaches that can either have fully observability, partial observability or no-observability. In the next sections we briefly describe each of those 5 types (Table 2.3).

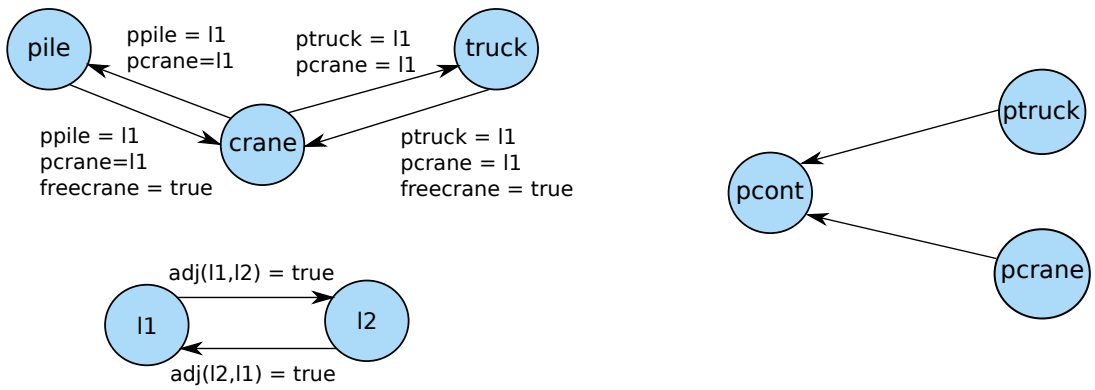


Figure 2.4: *Example of the domain transition graph and causal graph for two variables of the Dock worker robots: pcont (left top) and ptruck (left bottom), encoding the position of the container and of the truck, respectively; (right) causal graph for the variables of the problem.*

Observability \ Effects	Non-Deterministic	Probabilistic
Full $\forall s$	FOND	FOPP
Partial $\forall s$	POND-Contingent	POPP
Partial at s_0 / None $\forall s \neq s_0$	POND-Conformant	-

Table 2.3: *Different types of planning problems under uncertainty according to the assumptions they make about the world observability and the uncertainty on the effects of the actions.*

2.2.1 Non-Deterministic Planning

When planning with non-deterministic actions, the **solution** is a *policy*, i.e. a partial function mapping states into actions $\pi : S \rightarrow \mathcal{A}$, such that for every reached state this policy returns an action (except for the goal states, in which a policy returns no action or a NO-OP action). Non-deterministic planning problems can have three types of solutions:

- a ***weak*** policy, with no guarantees to achieve a goal state;
- a ***strong*** policy, which guarantees to achieve a goal state; and
- a ***strong cyclic*** policy that eventually guarantees to achieve a goal state, despite the cycles.

Note that a weak policy implies the existence of dead-end states, i.e., a state from which is not possible to reach a goal state.

Fully Observable Non-Deterministic Planning Problems (FOND)

In a Fully Observable Non-Deterministic planning problem (FOND) the state is fully observable (at s_0 and after every action), but there is uncertainty over the effects of the actions, i.e., actions can have more than one possible outcome, but it is not possible to estimate which effect will occur after executing an action, since there is no probability associated to them.

In theory a FOND problem can be solved by an AND/OR search in the state space, where the search branches: (*i*) on the OR nodes corresponding to alternative choices of actions; and (*ii*) on the AND nodes corresponding to all possible outcomes of an action. A solution for an AND/OR search is a DAG that only branches on the AND nodes; whereas the leaf nodes of the DAG are either goal states or dead-end states. If all the leaves are goal states the solution is strong or strong cyclic, and otherwise it is weak. However this is very inefficient. There are two main efficient approaches to solve FOND problems, that are:

- *Planning as Model Checking* (Cimatti et al., 2003) which describes the planning problem in a temporal logic languages and applies pre-image operations to find weak, strong-cyclic and strong policies.

- *Heuristic* AND/OR *search* (Muise et al., 2012) which finds a solution of a deterministic version of the problem for each branch of an AND node, and make backtracks to try to find first a strong policy; if it fails, it looks for a strong cyclic one, otherwise returns a weak policy.

Partially Observable Non-Deterministic Planning Problems (POND)

Some problems may present uncertainty not only in the effects of the actions, but also in the initial state, meaning the agent may not know the truth value of some facts of the world (Ghallab et al., 2004). We call this type of problems as Partially Observable Non-Deterministic Planning (POND). There exist two main types of POND problems, depending on the possibility to perform observations. If the agent is not allowed to perform observations, then we have a *conformant planning problem*; but if the agent is allowed to perform observations, then we have a *contingent planning problem*.

Conformant Planning (POND-Conformant). A *conformant planning problem* (Goldman and Boddy, 1996) is a POND where the agent is not able to perform observations to uncover the truth value of these facts along the visited states.

Formally, a conformant planning problem P is a classical planning problem, where the initial state I is no longer a single state but a set of possible states represented by a set of known fluents (either *True* or *False)*, that is the set of fluents whose value is known at the initial state I. This set of states deemed possible is called a *Belief State* (Bonet and Geffner, 2001). Since there is uncertainty in the initial state, the *Closed World Assumption* does no longer holds, because what is not present in the initial state could be *True* or *False*. Akin to the classical planning, the solution to a *conformant planning problem* is also a plan, i.e. an ordered sequence of actions, that must be applied in *every possible* initial belief state and reach a goal belief state. Due to the size of the belief state, and that *all* actions must be applicable in *all* states, finding a solution for a conformant planning problem is a hard task.

There are two main approaches to solve conformant planning problems:

- *Search in the belief state space*. In this search, an action is applicable in a belief state b_i if it is applicable in each (physical) state $s \in b_i$. Also, the non-deterministic actions are used to generate a single successor belief state. *Conformant-FF* planner (Hoffmann and Brafman, 2006) is an example of this solution.

- *K-Translation*, which translates the conformant problem to a deterministic planning problem described in an epistemic language using a state-of-the-art classical planner (Palacios and Geffner, 2007b).

Contingent Planning (POND-Contingent). A contingent planning problem is a POND where the agent is able to perform (partial observations) after applying an action, and then it can discriminate among different possible states, decreasing its uncertainty about the world.

Thus, a *contingent planning problem* P is a POND that has a set of possible initial states, i.e the initial *Belief State*, and a set of observations $\mathcal{O}$ that allows the agent to observe the truth value of a fluent $f \in F$. These observations divide the belief state in two different belief states, one in which the observed fluent holds and one in which it does not hold (an agent perform enough observations until there is no uncertainty). The solution of a *contingent planning problem* is a policy over the belief state space.

There are two main approaches to solve a contingent planning problem:

- *Contingent Heuristic* AND/OR *search* in the space of belief states. In this search , as for the conformant planning an action is applicable in a belief state b_i if it is applicable $\forall s \in b_i$, however the successor states are not combined into a simple successor belief state but it branches (AND node) to all possible outcomes and/or observations. An example is the *Contingent-FF* planner (Hoffmann and Brafman, 2005).

- *K-Translations*, which translates the contingent problem to an epistemic equivalent FOND problems and solves it using a state-of-the-art FOND planner (Albore et al., 2009).

2.2.2 Probabilistic Planning

In probabilistic planning, a probability can be associated to each outcome of a non-deterministic action reflecting how likely is for this effect to occur. Probabilistic planning problems can have two types of solutions:

- a ***proper*** policy, that is, a policy that reaches the goal with probability 1; and

- a ***improper*** policy, that reaches the goal with probability less than 1.

Fully Observable Probabilistic Planning (FOPP)

Probabilistic planning problems are commonly modeled as a Markov Decision Process (MDP) (Puterman, 1994), and inherit solutions for this area that optimize the expected cumulative costs of a policy execution, where at each step the state is fully observable.

An interesting subclass of MDPs, is called Stochastic Shortest Path MDPs (SSP-MDPs) (Bertsekas and Tsitsiklis, 1991) and will be explained in more details in Chapter 7. An SSP makes the assumption that there is a proper policy for every $s \in S$ and that every improper policy has an infinite expected cumulative cost. Solutions for probabilistic planning problems are mainly based on:

- *Synchronous Dynamic Programming* approaches, where in each iteration *all* states have its value function updated. For example, the algorithm *Value Iteration* (Puterman, 1994).

- *Asynchronous Dynamic Programming* approaches, where in each iteration only for a subset of states, generally those with more probability to be visited have its value function updated. Algorithm LRTDP (Bonet, 2003) is an example of this technique.

Partially Observable Probabilistic Planning (POPP)

Partially Observable Probabilistic Planning are a generalization of Fully Observable Probabilistic Planning and are modeled as Partially Observable Markov Decision Process (POMDP) where states are partially observable. POMDP can be understood as an MDP over belief states, where belief states, unlike in non-deterministic planning problems, are **probability distributions** over the possible states. The initial situation is characterized by an initial belief b_0, and an observation returns the probability $P_a(o|s)$ of o being true in state s.

Solutions for POMDP are mainly based on linear programming and point-based value updates (Shani et al., 2013).

2.3 Discussion about this chapter.

In this chapter we have presented a brief review of classical planing problems, pointing out the assumptions it makes and showing some planning systems (planners) to solve these problems. We have also seen how to relax those assumptions and obtain more realistic problems, when considering non-deterministic or probabilistic effects, and partial observation.

However, those problems may present dead-ends, that is, states from where there is no solution, and the only way to find a solution is to ask human help. In the next Part will look at non-deterministic planning problems (FOND) with full or partially observability, and introduce these new models formally presenting its different solutions, and how dead-ends may affect the search for a solution.

Part I

Human Help in Non-Deterministic Planning

In the previous chapter, we present a brief review on automated planning, defining classical planning problems and how it can be described using planning domain description languages such as STRIPS and SAS+. We also classified non-deterministic planning approaches into four different classes called: FOND, POND, FOPP and POPP.

In the next four chapters, we investigate how to use human help in planning with non-deterministic effects, that is, problems where there exist uncertainty in the effects of the actions, as well as partial observability.

We start in Chapter 3 by formally explaining: (*i*) fully observable non-deterministic problems (FOND) (Section 3.1.1); (*ii*) how to solve them (Section 3.1.2); and (*iii*) how dead-ends make these problems more difficult to solve (Section 3.1.2). Next, we present partially observable non-deterministic problems (POND) (Section 3.2), conformant (Section 3.2.1) and contingent planning problems (Section 3.2.2). We also present a new categorization of dead-ends, taking into account how the uncertainty in the initial situation may cause them, and show examples of them (Section 3.2.3). Solutions for POND problems are presented in Section 4, including a review of existent solutions and describing in details the k-translation approach to translate a conformant or contingent planning problem into a full-observable planning problem, as used in this thesis.

Then, in Chapter 4 we introduce human help (Sections 4.1 and 4.2), describing how can be used to obtain strong solutions for every one of these classes of dead-ends (Section 4.3). We also formally define the *domain transition graph*, that captures the conditions under which a fluent may change its value; and the *causal graph*, that shows the dependencies between the literals of the problem (Section 4.4). We will also show how to compute a relevant set of literals to the goal (Section 4.4.2), that can be used to create the human actions, as well as how to extract more information from these graphs to create an even smaller set of relevant literals (Section 4.4.5).

Next, in Chapter 5, we describe the different domains used in our experiments (Section 5.1), and we compare our approach to the state-of-the-art in contingent planning (Section 5.2). We show experiments of problems with dead-ends and human help to validate empirically our proposal, and also evaluate the scalability (Section 5.3).

Finally, in Chapter 6 we review related work, like Symbiotic Autonomy, and other solutions for contingent planning with dead-ends.

Chapter 3

Non-Deterministic Planning: background

In this chapter we present the foundations of non-deterministic planning. We start by describing FOND planning, i.e., Fully Observable Non-Deterministic planning, that relaxes the assumption of deterministic effects of classical planning but preserves the full observability. Next, we show one solution of FOND problems with dead-ends and describe the state-of-the-art FOND planner, called PRP. Then, we present the foundations of POND planning, i.e., Partially Observable Non-Deterministic planning, usually referred in the literature as conformant or contingent planning. Then we present an approach, considered the state-of-the-art, to solve POND planning problems that translates a POND problem to a FOND problem to be solved with the PRP planner.

3.1 Planning with fully observable states and non-deterministic effects

As we have discussed in Section 2.2 non-deterministic planning can be divided in two main classes: *fully observable non-deterministic planning* (FOND) and *partially observable non-deterministic planning* (POND). In a FOND planning problem the agent has full information about the initial state and after executing a non-deterministic action, the agent has full observability of the resulting state.

Example 2 (Triangle Tireworld Domain)**.** *In the Triangle Tireworld domain, a car driven by an agent moves through connected locations in order to reach a goal location. At each move between locations the car can have a flat tire. Some locations contain a spare tire, however, the agent would be in a dead-end if it has a flat tire in a location with no spare.*

The non-determinism in this domain arises from the non-deterministic effect of having a flat tire or not, after every move action. Figure 3.1 shows an example of a FOND planning problem in the Tireworld domain, where gray nodes are locations that have spare tires; the initial location is loc_{11}

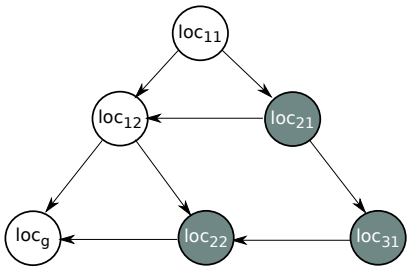


Figure 3.1: *Example of a Triangle Tireworld map, with six locations, where loc_g is the goal location, and loc_{11} is the initial location. Directed edges indicate connected locations. Gray and white nodes indicate, respectively, locations with and without spare tires.*

π_1: strong policy	π_2: weak policy
$(loc_{11}, no\_flat, no\_spare) : moveto(loc_{21})$ $(loc_{21}, no\_flat, spare) : moveto(loc_{31})$ $(loc_{31}, no\_flat, spare) : moveto(loc_{22})$ $(loc_{22}, no\_flat, spare) : moveto(loc_g)$ $(loc_{21}, flat, spare) : changetire()$ $(loc_{31}, flat, spare) : changetire()$ $(loc_{22}, flat, spare) : changetire()$	$(loc_{11}, no\_flat, no\_spare) : moveto(loc_{12})$ $(loc_{12}, no\_flat, no\_spare) : moveto(loc_g)$ $(loc_{12}, flat, no\_spare) : -$

Table 3.1: *Examples of policies for the* FOND *planning problem in the Triangle Tireworld problem of Figure 3.1. (left) An example of strong solution policy for each of the 7 reachable states. (right) An example of weak policy that can lead to a dead-end state.*

and loc_g is the goal location. The agent must reach the goal traversing the connected locations. The solution is a strong policy that guarantees to reach the goal (Table 3.1 (left)). The solution π_2 (Table 3.1 (right)) is a weak policy that may lead the agent to the goal, with a possibility of reaching a dead-end, i.e., the agent can end up in a location loc_{12} with flat tire and no spare.

3.1.1 FOND planning problems

Definition 6 (FOND planning model)**.** *Formally, a* FOND *planning model is a tuple* $\mathcal{M}^{\text{FOND}} = \langle S, s_0, S_G, \mathcal{A}, \mathcal{T}, C \rangle$*, where* S*,* S_0 *and* S_G *are as defined in Definition 1, and:*

- $\mathcal{A}$ *is a set of non-deterministic actions, where* $\mathcal{A}(s) \subseteq \mathcal{A}$ *denotes the set of actions applicable in the state* $s \in S$*;*
- $\mathcal{T} : S \times \mathcal{A} \to 2^S$ *is a non-deterministic transition function over states, where* $\mathcal{T}(s, a)$ *indicates a non-empty set of successors states of* s*, obtained applying the action* $a \in \mathcal{A}(s)$ *in the state* $s \in S$*; and*
- $C : \mathcal{A} \to \mathbb{R}^+$ *is a cost function, where* $C(a)$ *denotes the cost of applying action* $a \in \mathcal{A}$*; terminal states that are goal states have a fixed cost* $c_p = 0$ *(independent of the actions) and terminal states that are dead-ends have a cost* $c_p \neq 0$*.*

◇

A non-deterministic state model $\mathcal{M}^{\text{FOND}}$ as in Definition 6 can be represented as an AND/OR graph, where states correspond to OR nodes (with alternative action choices) and the actions correspond to AND nodes (with the conjunction of non-deterministic effects). Figure 3.2 shows an example of an AND/OR graph, where s_i are OR nodes and $\square$ are AND nodes.

In the set-theoretic STRIPS (Section 2.1.2) extension for FOND planning, a FOND problem is given by:

Definition 7 (Set-theoretical FOND planning problem)**.** *A* FOND *planning problem is the tuple* $\mathcal{M}^{\text{FOND}}_{\text{STRIPS}} = \langle F, I, Op, G \rangle$*, where* F*,* I *and* G *are as in Definition 2 and:*

- Op *is the set of operators, where each non-deterministic action* $a \in Op$ *is given by a tuple* $\langle prec(a), eff(a), cost(a) \rangle$ *where:* $prec(a)$ *is a list of preconditions that must be true in the state before applying the action,* $eff(a)$ *is a set of effects, and* $cost(a)$ *of preconditions, a set of effects and an integer representing the action cost. The effects* $eff(a)$ *is a set of pairs* $(\langle add(e_1), del(e_1) \rangle, ..., \langle add(e_n), del(e_n) \rangle)$ *where each pair indicates the non-deterministic effect of action* a*, given by an add and delete list of propositions.*

◇

Definition 8 (FOND Policy)**.** *A (stationary)* ***policy*** *for a* FOND *problem is a mapping from states to actions,i.e.,* $\pi : S \to A$*, prescribing the action* $\pi(s)$ *that must be taken in the state* s*. A policy can be complete if it is defined* $\forall s \in S$ *or partial, if it is defined* $\forall s \in S' \subset S$*. A* FOND *policy can be one of the three types: weak, strong or strong cyclic.* ◇

Definition 9 (History)**.** *A history* $h = \langle s_0, s_1, \ldots, s_{|h|} \rangle$ *is a sequence of states visited by the agent, when following a policy* π*.* ◇

Let the set of all histories of a FOND problem $\mathcal{M}^{\text{FOND}}$ induced by a policy π be denoted as $\mathcal{H}^{\pi}$. Since a history can end up either in a goal state, or in a dead-end state, a policy can be classified in one of the three types:

- Weak: if there exists at least one history $h \in \mathcal{H}^{\pi}$ ending in a goal state;
- Strong: if all histories end in a goal state, i.e., $\forall h \in \mathcal{H}^{\pi}, s_{|h|} \in S_G$; and all histories do not visit the same state twice; and
- Strong cyclic: if all histories end in a goal state, i.e., $\forall h \in \mathcal{H}^{\pi}, s_{|h|} \in S_G$; and there exists at least one history where a state is visited twice.

We can classify FOND problems in three classes: FOND with no dead-ends, FOND with *avoidable* dead-ends and FOND with *unavoidable* dead-ends. A FOND problem with no dead-ends has only strong (cyclic) policies $\forall s \in S$. A problem with *avoidable* dead-ends has at least one strong (cyclic) policy rooted in the initial state, and a FOND problem with *unavoidable* dead-ends has only weak policies. In this work we make the following assumption:

Assumption I: there is always **at least a weak policy from the initial state**, i.e., there is at least a path from the initial state to a goal state.

3.1.2 Solving a FOND Planning Problem

In this section we present different optimization criteria for FOND planning. We first present solutions for FOND problems with no dead ends, followed by solutions for FOND problems with avoidable and unavoidable dead-ends.

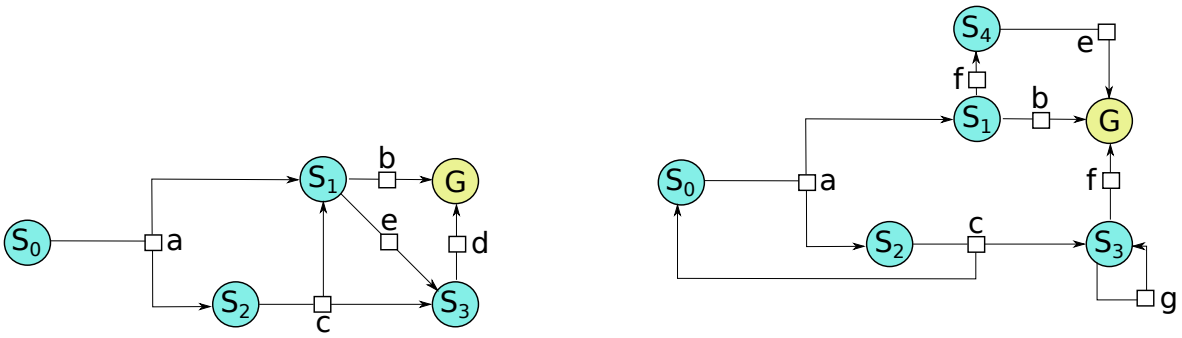


Figure 3.2: *Two examples of* FOND *planning problems with no dead-ends. (left)* FOND *planning problem with 5 states and no cycles; (right)* FOND *planning problem with 6 states and cycles. G stands for the goal state*

FOND problems with no dead-ends

Given a FOND problem $\mathcal{M}^{\text{FOND}}$ (Definition 6) with no dead-end states, any policy for $\mathcal{M}^{\text{FOND}}$ is a strong (cyclic) policy. Figure 3.2 shows two examples of FOND problems with no dead-ends: (left) FOND with no cycles, and (right) FOND with cycles.

The solution of a $\mathcal{M}^{\text{FOND}}$ problem with no dead-ends can be expressed in terms of the well-known Bellman equation (Bellman and Kalaba, 1957) that characterizes the *optimal value function* $V^*(s)$ (Bonet and Geffner, 2005a; Geffner and Bonet, 2013):

$$V^*(s) \stackrel{def}{=} \begin{cases} 0, & \text{if } G \subseteq s, \\ min_{a \in A(s)} Q^*(a, s), & \text{otherwise,} \end{cases} \tag{3.1}$$

where $Q^*(a, s)$ is the optimal accumulated cost for applying action a in state s, according to an utility function defined for the pair state-action. Thus, choosing which is the best action to apply at state s depends on the utility function used to aggregate the (conjunctive) non-deterministic effects of action a. Bonet and Geffner (2005a) only defined two utility criteria to aggregate action effects: *max* (pessimistic) (Equation 3.2) or *add* (overall) (Equation 3.3):

$$Q^*(a, s) = \mathcal{C}(a) + max_{s' \in \mathcal{T}(a,s)} V^*(s') \ (max), or \tag{3.2}$$

$$Q^*(a, s) = \mathcal{C}(a) + sum_{s' \in \mathcal{T}(a,s)} V^*(s') \ (add). \tag{3.3}$$

Also we introduce the *min* (optimistic) criterion (Equation 3.4), since it is used by the non-deterministic planner PRP:

$$Q^*(a, s) = \mathcal{C}(a) + \min_{s' \in \mathcal{T}(a,s)} V^*(s') \ (min). \tag{3.4}$$

The optimal policy π^* returns the best action to apply in each state s, according to the utility functions of Equations 3.2, 3.3 or 3.4, i.e. the action that minimizes the Equation 3.1:

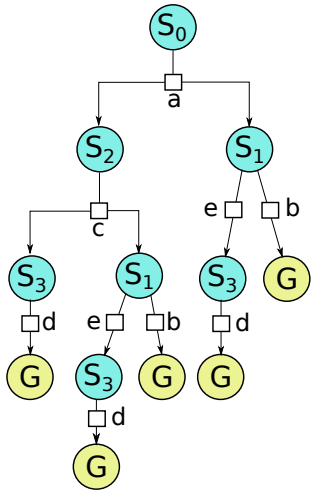
$$\pi^*(s) = \underset{a \in A(s)}{argmin} \ Q^*(a, s), \forall s \in S \tag{3.5}$$

The *max* pessimistically considers that every time a non-deterministic action a is applied, the outcome of action a will be the state with the maximum value, which is a robust solution (i.e. the best choice in the worst case). The *add* criterion considers the sum of the values of all outcomes, which can choose the action with the best overall (expected) outcomes. The *min* criterion is optimistic, because it (optimistically) considers the outcome of action a will be the state with the minimal value. To illustrate these optimization criteria and their differences, consider the FOND problem of Figure 3.2 (left) with no cycles and its corresponding AND/OR graph and value function (Figure 3.3). The values of each state are computed having the optimal value of the goal state $V^*(G) = 0$. E.g., when applying Equation 3.1 in state s_3, the value $V^*(s_3)$ is the cost of action b plus the $V^*(G)$, i.e., $V^*(s_3) = \mathcal{C}(b) + V^*(G) = 1$. Likewise, for state s_3 and action d, $V^*(s_3) = 1$.

The optimal value of s_2 is the minimal of $Q^*(e, s_1)$ and $Q^*(b, s_1)$, which is 1. To compute the values of s_0 and s_2, we must select one of the aggregation criteria: *min*, *max* or *add*. Figure 3.3 (right) shows a table with V^* for all states with the three criteria.

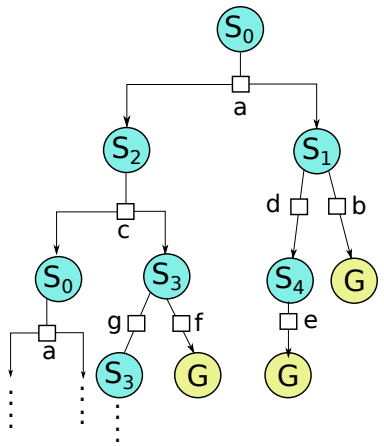
When considering problems with cycles, the value of the states can increase to ∞ when considering the *max* and *add* criteria. In fact, Equations 3.2 and 3.3 will sum action costs until ∞. But the *min* criteria will consider only the values of finite paths. To illustrate this, consider the FOND planning problem depicted in Figure 3.2 (right) (for which there are only strong cyclic policies) and its corresponding AND/OR graph and value function (Figure 3.4). In this problem, the value of applying action c in state s_2 using the *max* criterion will choose the maximum between $V^*(s_0)$ and $V^*(s_3)$ the accumulated value $Q^*(c, s_2)$, that is the outcome that results to the state s_0, that in turn depends on s_2, creating a cycle of transitions that will keep increasing, and thus $Q^*(c, s_2) = \infty$. This cycle is represented by a branch on the graph, depicted in Figure 3.4 (left), that can grow indefinitely until ∞. The optimal value function V^* for all the states of the FOND problem in Figure 3.4 (right), using these criteria, are shown in the table in Figure 3.4 (right). Notice that, the only criteria that computes a finite value for all states is *min*.

Thus, given two policies π_1 and π_2, for a FOND problem with cycles, they are comparable only



	V^π		
	min	**max**	**add**
G	0	0	0
s_0	2	3	5
s_1	1	1	1
s_2	2	2	3
s_3	1	1	1

Figure 3.3: *(Left)* FOND *planning problem with 5 states and no dead-ends. State space for a fully observable non-deterministic (*FOND*) planning problem. G stands for the goal state. (Right) Accumulated values for each state according with the three different criteria: min, max and add.*



	V^π		
	min	**max**	**add**
G	0	0	0
s_0	2	∞	∞
s_1	1	1	1
s_2	2	∞	∞
s_3	1	1	1

Figure 3.4: *(Left) State space for a fully observable non-deterministic (*FOND*) planning problem with cycles. (Right) Cumulative values for the different states of the* FOND *planning problem in figure 3.3 (left), applying the three different criteria.*

using *min* criterion, since *max* and *add* will return ∞ due to the infinite possible histories.

FOND problems with dead-ends

In these problems we also want to find a policy that minimizes the expected accumulated cost. Hence, th optimal solution of a $\mathcal{M}^{\text{FOND}}$ problem with dead-ends can be expressed in terms of the *optimal value function* $V^*(s)$:

$$V^*(s) \stackrel{def}{=} \begin{cases} 0, & \text{if } G \subseteq s, \\ \infty, & \text{if } s \text{ is a dead-end}, \\ min_{a \in A(s)} Q^*(a, s), & \text{otherwise.} \end{cases} \tag{3.6}$$

Note that when considering FOND problems with dead-ends, if the value of a state s is ∞ either the state s leads to a dead-end or s leads to a cycle. Figure 3.5 shows two examples of FOND problems with avoidable (left) and unavoidable (right) dead-ends.

FOND problems with avoidable dead-ends. In this case, the *min* criterion may fail to select a strong (cyclic) policy because when computing the accumulated cost throughout a non-deterministic

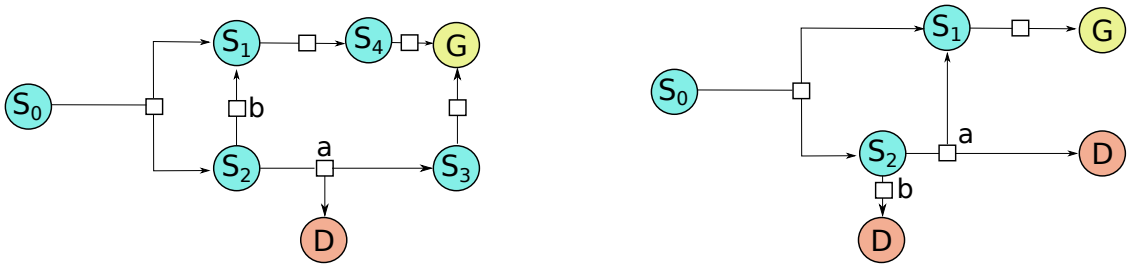


Figure 3.5: *Two examples of* FOND *problems with dead-ends. (left)* FOND *problem with an avoidable dead-end; (right)* FOND *problem with unavoidable dead-ends.*

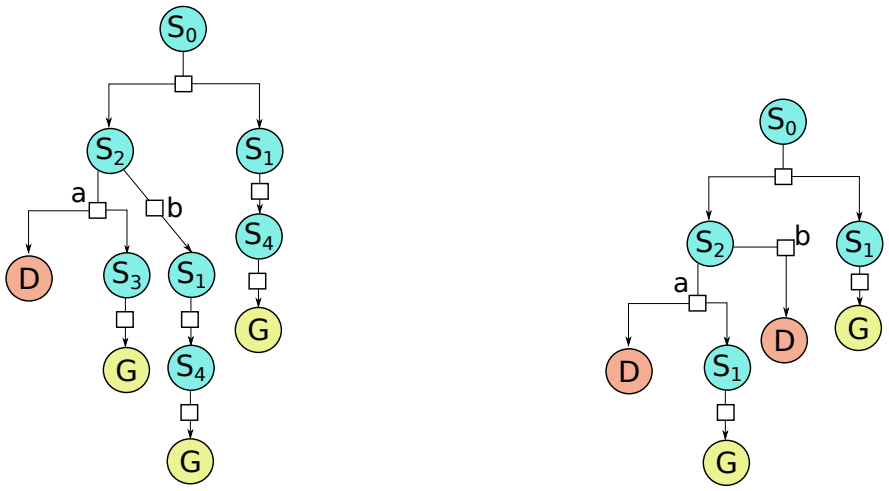


Figure 3.6: *Two graphs for the* FOND *problems with dead-ends of Figure 3.5. (left)* FOND *problem with an avoidable dead-end; (right)* FOND *problem with an unavoidable dead-end.*

action a, that can lead to a dead-end, it only consider the outcome with finite value. On the other hand, a planner using *max* or *add* criterion will correctly detect and penalize actions that lead the agent towards dead-ends. Figure 3.6 (left) shows the graph for the FOND problem depicted in Figure 3.5 (left). Notice that the *min* criterion may select a weak policy, instead of the strong (cyclic) for s_2, since $V^*(s_3) < V^*(s_1)$. If we assume there are no cycles, *max* and *add* criteria will recognize the infinite value $V^*(D) = \infty$ and select to expand the action that leads to s_1.

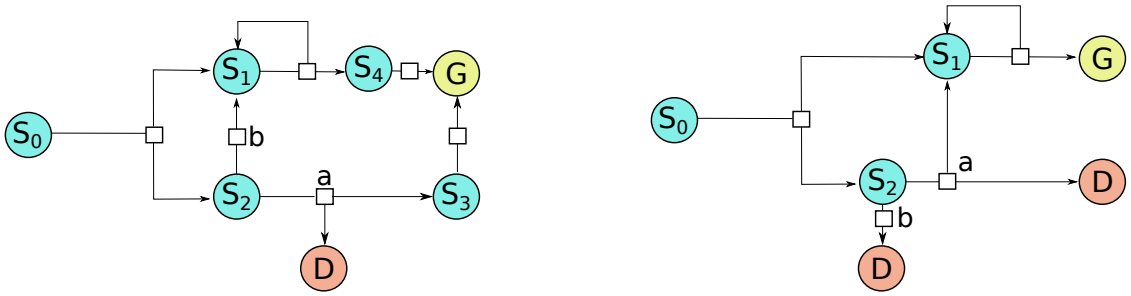


Figure 3.7: *Two examples of* FOND *problems with dead-ends. (left)* FOND *problem with an avoidable dead-end; (right)* FOND *problem with unavoidable dead-ends. Notice that there exist cycles.*

Now, if we assume there exist cycles (Figure 3.7 and Figure 3.8, $V^*_{max}(s_1) = V^*_{max}(s_3) = \infty$, hence both policies are not comparable. The same happens with the *add* criterion.

In sum, given two policies π_1 and π_2, for a FOND problem with avoidable dead-ends, they cannot be compared (with or without cycles) using the *min* criterion; but using *max* or *add* criterion, π_1 and π_2 can be compared only if there are no cycles.

Proposition 1. *For a* **FOND** ***planning problem with avoidable dead-ends****, min criterion may*

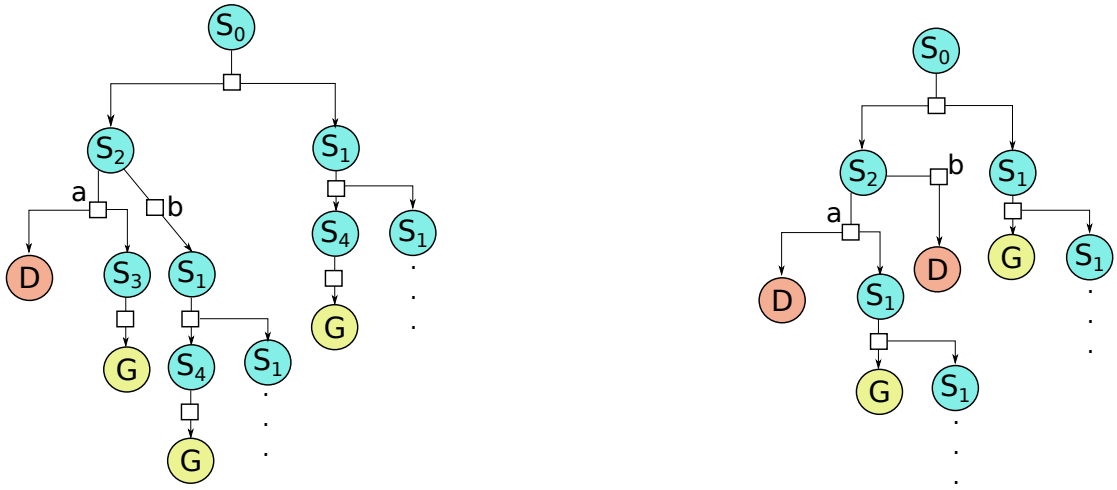


Figure 3.8: *Two graphs for the* FOND *problems with dead-ends of Figure 3.5. (left)* FOND *problem with an avoidable dead-end and a cycle; (right)* FOND *problem with unavoidable dead-ends and cycles.*

fail to find a strong (cyclic) policy.

FOND problems with unavoidable dead-ends. In this case, and under Assumption 1, the *min* criterion can be used to select the policy with the shortest history to the goal, independent of cycles. However, a planner using the *max* or *add* criteria will fail to compare policies, because they will assign ∞ to s_0 for all (weak) policies $V^*_{max}(s_0) = V^*_{add}(s_0) = \infty$. Figure 3.6 (right) shows the graph for the FOND problem depicted in Figure 3.5 (right). Notice that the *min* criterion will select an action a, even considering with cycles with unavoidable dead-ends, since $V^*_{min}(s_1) = 1$, while $V^*_{max}(s_1) = \infty$.

Given two policies π_1 and π_2, for a FOND problem with unavoidable dead-ends, and assuming there is a path to the goal, π_1 and π_2 can be compared only under the *min* criterion, since *max* and *add* criteria will not distinguish among cycles and dead-ends, returning ∞ for both cases.

Proposition 2. *For a* **FOND** ***planning problem with unavoidable dead-ends****, the min criterion can be used to find the weak policy that induces the history to the goal with the minimum cost.*

3.1.3 An efficient planner for FOND problems: PRP planner

As we have discussed in the previous section, there is not a clear optimization criterion for FOND problems with dead-ends and cycles. Thus, solvers of non-deterministic planning problems usually use heuristic methods that transform the original problem containing the set of non-deterministic actions A into a *determinized* version A', where every action $a \in A'$ is deterministic. The all outcomes determinization creates the set A' by defining a new action for every outcome of a non-deterministic action. Notice that the optimal solution for the all outcomes determinization is also the policy that satisfies the *min* criterion.

The PRP (*Planner for Relevant Policies*) (Muise et al., 2012), considered the state-of-the-art FOND planner, returns a strong plan π if one exists. If there is not a strong policy, PRP returns a strong cyclic policy, otherwise it returns a weak policy. PRP solves a FOND problem $\mathcal{M}^{\text{FOND}}$ by solving several deterministic planning problems (generated with a determinization of $\mathcal{M}^{\text{FOND}}$) and then returning a policy that is a composition of these deterministic solutions. While the policy π_{FOND} for the original problem $\mathcal{M}^{\text{FOND}}$ must be defined for all states reachable from s_0, the deterministic solutions π_{DET} are only defined over a subset of S. The policy π_{FOND} is initialized mapping all states with *noop* (dummy) actions. This process is done in two phases:

(1) **Search**: PRP computes a solution π_{DET} for a determinized version of the problem (using an efficient classical planner) and then, it updates π_{FOND} for every state of π_{DET}, i.e., $\pi_{\text{FOND}}(s) = \pi_{\text{DET}}(s)$.

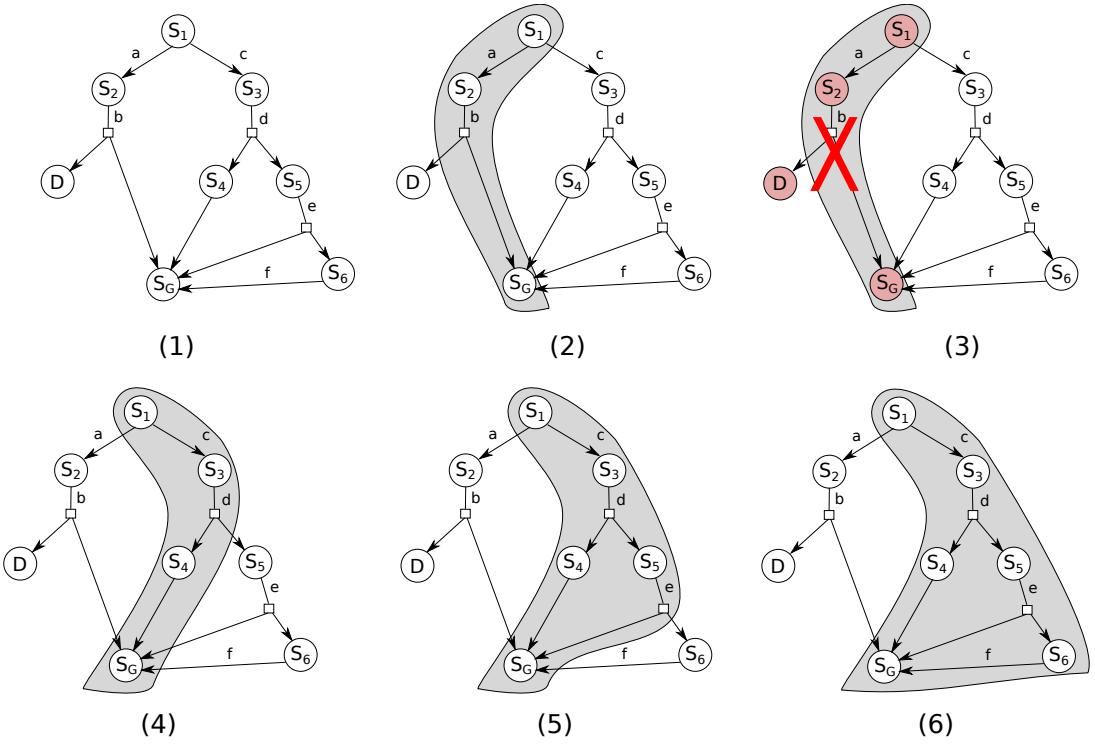


Figure 3.9: *Example of a* PRP *execution: (1) the original* FOND *problem with goal s_G and dead-end D; (2) First phase of the algorithm, the determinized solution is marked in light gray; (3) In the completion phase a dead-end state is found and the pair (s_2, b) is marked as forbidden, i.e., b is removed from $A(s_2)$; (4) A new determinized solution is found from s_0 avoiding the actions marked as forbidden; (5) and (6)* PRP *completes updating the policy π_{FOND} by searching for determinized solutions from all reachable states from s_0.*

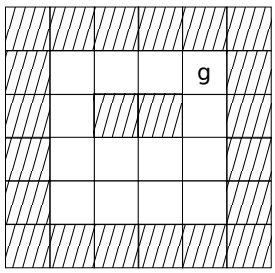
(2) **Completion**: PRP analyze all outcomes of the actions of π_{FOND} starting from the initial state, until reaching an outcome state s such that $\pi_{\text{FOND}}(s) = \mathit{noop}$, and then it repeats (1) from state s.

If during the completion, a dead-end is encountered, as the effect of an action a chosen in state s, PRP stores the pair (s, a) in a set of forbidden *state-action* pairs. In this case, the policy π_{FOND} computed so far is initialized again with *noop* actions for all states, and the search is restarted from scratch, but knowing now the forbidden state-actions pairs that lead to dead-ends.

Another technique used by PRP is to work with partial states, returning a more general policy and focusing on the relevant part of a state that can lead to the goal.

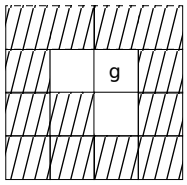
To understand the PRP algorithm consider the FOND example showed in Figure 3.9 (1). PRP starts by determinizing the original problem with the *all-outcomes* determinization . Then, PRP finds a solution for this determinized problem and update the policy π associating the state with the corresponding action of the determinized solution $\{(s_1, a); (s_2, b)\}$ (Figure 3.9 (2)); PRP then tries to complete the policy π_{FOND} computed so far. When looking for all outcomes of action b in s_2, it realizes that one outcome is a dead-end. The non-deterministic action b is then stored along with the state s_2 as a forbidden pair (Figure 3.9 (3)). Then PRP restarts the whole process considering now the set of forbidden pairs which results in the policy π_{DET} of Figure 3.9 (4); then PRP continues to compute the solution for the remaining states, until it finishes returning the policy $\{(s_1, c); (s_3, d); (s_5, e); (s_6, f)\}$.

PRP is capable of detecting dead-ends during the search and tries to return a strong policy first, then if one does not exist, it tries to return a strong cyclic policy. When there is no strong (cyclic) solution, the weak policy returned by PRP can contain multiple branches ending in a dead-end, however, PRP does not guarantee to return a weak policy containing the shortest path to the goal (Muise et al., 2012). PRP can also be configured to solve problems with non uniform action costs.



Conformant Plan:
E,E,E,E,N,N,N,N,E,E,E,E

Figure 3.10: *Example of a **conformant planning problem** in the Localize domain. (left) A 6×6 grid world, where g is the goal location, the hatched cells are walls and the initial state is any of the free locations. (right) A conformant plan solution: a sequence of actions with no observations that guarantees to reach the goal.*



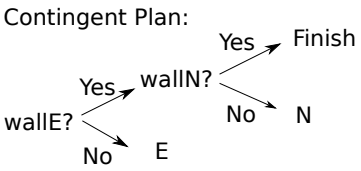


Figure 3.11: *Example of a **contingent planning problem** in the Localize domain. (left) A 4×4 grid world, where g is the goal location and the hatched cells are walls. (right) The contingent solution: a conditional plan that branches on the result of the observations that guarantees to reach the goal (strong policy).*

3.2 Partial Observability and Non-Determinism (POND)

Partially Observable Non-Deterministic (POND) planning is usually referred to in the literature as *conformant planning* or *contingent planning* (Section 2.2). Due to the partial observability, the initial state is uncertain and can be represented by a set of possible initial states b_0 (an abstract state), i.e., the agent may be in one of the states $s \in b_0$. In conformant planning (Goldman and Boddy, 1996) there are no observations, and the agent must create a plan that *conforms* to every possible initial state s_0 while in contingent planning problems the agent performs observations after each action.

Example 3 (Localize Domain)**.** *In the Localize domain an agent must reach a goal location in a grid world, where some cells represent walls, without knowing its initial location. The agent is able to move in any of the four directions (N, S, E and W), and stays in the same state when it moves towards a wall. Since the agent does not know its initial situation, the solution must satisfy the goal for **all possible initial states**.*

Figure 3.10 (left) shows an example of conformant planning problem in the Localize domain. The agent is in a 6×6 grid world, does not know its initial location, and it is not allowed to make observations. The solution is a plan, a sequence of actions that guarantees to reach the goal, i.e., it conforms to any possible initial location of the agent. Figure 3.11 (left) shows an example of contingent planning problem in the Localize domain (Example 3). The agent is in a 4×4 grid world, and does not know its initial location. But unlike in conformant planning, the agent is allowed to perform observations. The agent is able to sense if there is a wall in each of the four directions (*wallN?*, *wallS?*, *wallE?* and *wallW?*) and can move in any of the four directions (N, S, E and W). The solution is a conditional plan branching on the observations.

An approach to solve a conformant planning problem is a search in the belief state space where for each non-deterministic action the successor state is a single belief state. Thus the solution is a totally ordered sequence of actions (e.g. Figure 3.10 (right)).

An approach to solve a contingent planning problem is a search in an AND/OR belief state space,

where an OR node is a state for which we want to choose an action and an AND node lead to states (branchs) that must be necessarily part of the policy (corresponding to non-deterministic outcomes of an action or observation). The solution is then a conditional plan branching on the different outcomes of the observations or the non-deterministic actions (e.g. 3.11 (right)).

In the next sections we will show the formalization of conformant and contingent planning problems.

3.2.1 Conformant Planning

Definition 10 (Conformant planning model)**.** *(Palacios, 2009) The conformant planning model is a tuple* $\mathcal{M}^{\text{CONF}} = \langle S, S_0, S_G, A, \mathcal{T}, C\rangle$*, where* S*,* S_G*,* A *and* C *are the same as in Definition 6 and:*

- S_0 *is a non-empty set of possible initial states* $S_0 \subseteq S$*, and*
- $\mathcal{T}$ *is a non-deterministic transition function over belief states* $\mathcal{T} : 2^S \times \mathcal{A} \rightarrow 2^S$*, where* $b' = \mathcal{T}(b, a)$ *indicates the non-empty set of belief successors states denoted by* b'*, obtained applying action* a *in the belief state* b*. An action* a *is applicable in a belief state* b *iff* a *is applicable* $\forall s \in b$*, i.e.,* $a \in A(s), \forall s \in b$*.*

◇

Notice that, unlike fully observable planning, the set of states S_0 is a belief state, i.e., a **set of possible initial states**, that will be called b_0. A conformant plan is a sequence of actions $a_1, a_2, ..., a_n$ that is applicable in every possible initial state $s \in b_0$ and every transition $\mathcal{T}(s, a)$, generating a sequence of belief states $b_0, b_1, ..., b_n$ such that the execution of action a_i in a belief state b_i result in the belief state $b_{i+1} \in \mathcal{T}(s_i, a_i)$, and where all the states $s \in b_{n+1} \subseteq S_G$.

Definition 11 (Set-theoretical Conformant planning problem)**.** *(Palacios, 2009) A conformant planning problem, in an extended version of STRIPS language, is* $\mathcal{M}^{\text{CONF}}_{strips} = \langle F, I, Op, G\rangle$*, where:*

- F *is the set of propositions of the problem and Lit is a set of literals over* F*, i.e.,* $l \in Lit$ *is a proposition in* F *or its negation;*
- I *is the set of* ***clauses*** *over Lit defining the initial belief state* b_0*; the non-unary clauses* $D \in I$ *are all invariants (called the set of axioms* $D \subset I$*) that also must hold for all successor states;*
- Op *is the set of operators, where each non-deterministic action* $a \in Op$ *is given by a tuple* $\langle prec(a), eff(a), cost(a)\rangle$ *of preconditions, a set of effects and an integer representing the cost, and the effects* $eff(a)$ *are a list of tuples* $(\langle add(e_1), del(e_1)\rangle, ..., \langle add(e_n), del(e_n)\rangle))$*; and*
- G *is a clause defined over fluents of* F *that defines the goal.*

◇

Due to the uncertainty, the *closed world assumption* (CWA) no longer holds, hence, a state s is a truth assignment to **all** literals in *Lit*. A literal l holds in a state s iff s assigns l to be true. We will often abuse notation and treat a set of literals as a conjunction.

An action is applicable in a belief state b if $prec(a)$ holds in $\forall s \in b$. The set $eff(a)$ is the set of effects of action a describing the possible outcomes of an action, where each effect $e \in eff(a)$ consists of a list of tuples $\{E_1, E_2, ..., E_n\}$ and each tuple $E_i \in eff(a)$ is composed of two lists of propositions $\langle add(e), del(e)\rangle$, expressing the add and delete list respectively. An effect $e \in eff(a)$ with more than one E_i called a non-deterministic effect. The successor belief state b' resulting of applying an action a in a belief state b is given by the following expression (Bonet and Geffner, 2011, 2014a):

$$b' = \mathcal{T}(b, a) = \bigcup_{s \in b} \Big[\bigcup_{e \in eff(a)} (s \setminus del(e) \cup add(e)) \Big]. \tag{3.7}$$

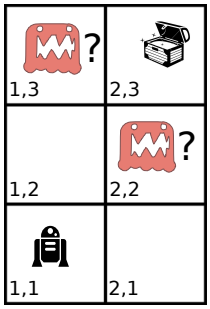


Figure 3.12: *Example of a contingent planning problem in the Wumpus World. The agent starts in location (1,1) and must reach the gold in location (2,3). There is a Wumpus either at (2,2) or (1,3).*

The solution of a conformant planning problem is a sequence of actions, that are applicable in the initial state S_0 and lead the agent to the goal. Since observations are not allowed in conformant planning, this sequence of actions must reach the goal regardless of the non-determinism of the actions.

3.2.2 Contingent Planning

Planning under uncertainty may be difficult without performing observations. Contingent planning models problems where the agent performs observations during the execution of the plan to possibly eliminate uncertainty.

Figure 3.12 shows an example of contingent planning problem, in a simplified version of the Wumpus World (Russell and Norvig, 2010), where the agent's goal is to reach the gold location and not be killed by a monster, called Wumpus. The uncertainty in this problem lies in the position of the Wumpus, partially known in the initial state. The agent, starting at location (1,1), can move in any direction inside the grid and sense if there is a smell that indicates the presence of the Wumpus in an adjacent cell; if there is no smell, all the adjacent cells are considered to be safe, so the agent can move to one of them. The actions are $move(loc_1, loc_2)$ and the observations are $sense(loc)$.

Figure 3.13 shows the AND/OR graph for the Wumpus World problem of Figure 3.12. The initial belief state is formed by the two possible initial states (i.e., each state representing a possible location of the Wumpus). As showed in the previous section, when the agent applies an action it receives an observation and its belief state is updated computing the next belief state considering every state of the previous belief state (Equation 3.7). Then, it is possible to split the belief states into those in which the observed fact holds, and those in which the observed fact does not hold.

Definition 12 (Contingent Planning Model.)**.** *(Albore, 2012) The contingent planning model is a tuple* $\mathcal{M}^{\text{CONT}} = \langle S, S_0, S_G, \mathcal{A}, \mathcal{O}, \mathcal{T}, C \rangle$*, where all elements are defined as in the conformant planning model (Definition 10), plus the observation function:*

- $\mathcal{O} : S \times \mathcal{A} \to 2^{\mathcal{L}}$,

mapping pairs state-action into sets of observable literals defined over a set of observable literals $\mathcal{L} \subseteq Lit$. ◇

The expression $l \in \mathcal{O}(s, a)$ means that the truth-value of l is observed when s is the real (physical) state of the world, and a is the last action applied.

Definition 13 (Contingent Planning Problem.)**.** *(Albore, 2012) A contingent planning problem in an extended version of* STRIPS *language is a tuple* $\mathcal{M}^{\text{CONT}}_{strips} = \langle F, I, Op, O, G \rangle$*, where* F, I, Op *and* G *are defined as in a conformant planning problem (Definition 11), with the addition of the set of observable literals:*

- O *is a set of conditional observations, where each* $o \in O$ *has a set of preconditions* $prec(o) \subseteq F$*, and when triggered by a pair* (s, a) *it uncovers the truth value of an observable literal*

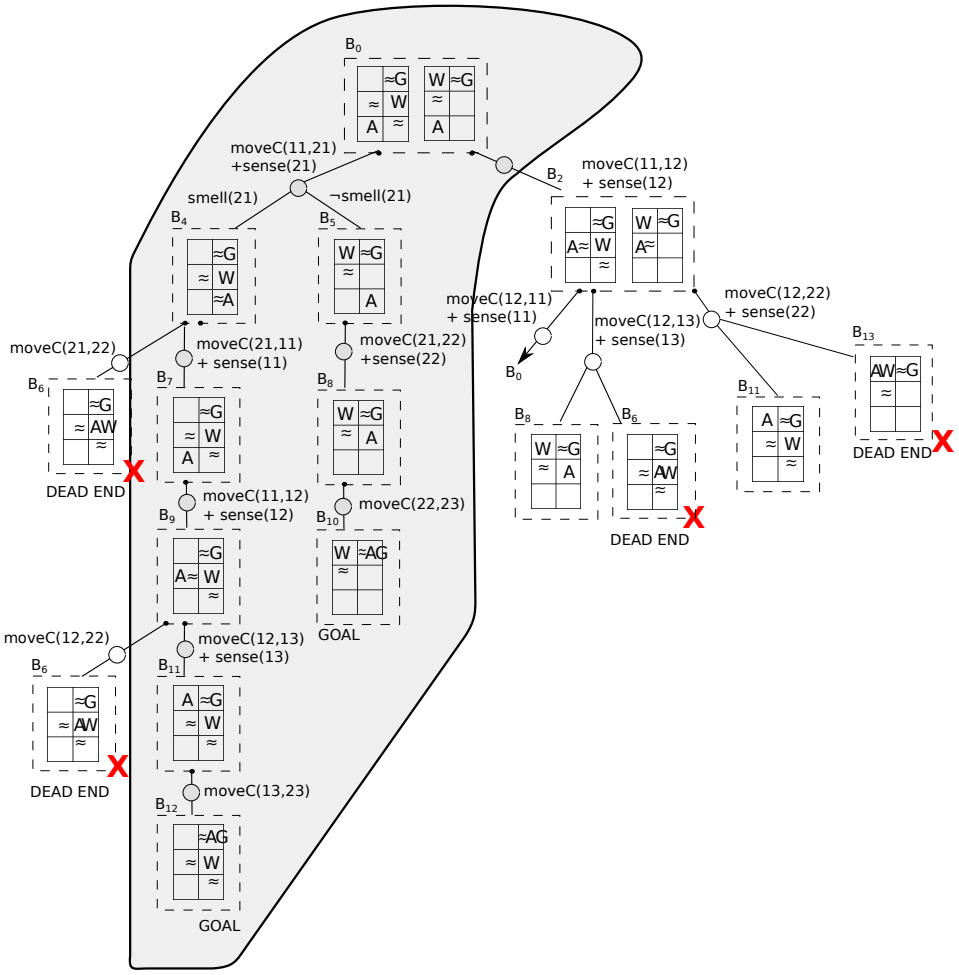


Figure 3.13: *Contingent planning in the Wumpus World: the belief space search, the agent starts at location (1,1) and must grab the gold at (2,3). Belief states b_i are represented by dashed boxes; Arrows indicate the result of actions and observations.*

$l \in \mathcal{O}(s,a)$, i.e., $o \in O$ is a pair $\langle prec(o), l\rangle$, where $prec(o) \subseteq F$ is the condition to trigger the observation of the truth value of the literal $l \in Lit$ of F. Thus, if an action $a \in A$ has an effect $prec(o)$, then the observation of l is triggered by action a.

◇

Thus, after executing an observation the belief state size will decrease. The process of updating the belief state b after performing action a, and receiving observation o, considering the set of axioms D, is given by the following equation (Bonet and Geffner, 2014a):

$$b'' = UPDATE(b', l) = UNIT(b' \cup l \cup D), \tag{3.8}$$

where b' is the state resulting from applying action a in the belief state b, as indicated in Equation 3.7, and $UNIT(s)$ is the operation that returns the set of states where the observed literal $l \in \mathcal{L}$ holds, and the non-unary clauses D in the initial state holds. Equation 3.8 is called *Belief tracking*.

As an example of a contingent planning problem, consider the Wumpus World problem showed in Figure 3.12. Propositions in F encode the different facts of the problem:

- $AgentAt(loc)$ indicates the position of the agent;
- $GoldAt(loc)$ indicates the position of the gold; and
- $WumpusAt(loc)$ indicates the position of the Wumpus (unknown in the initial state).

The agent action is $move(loc1, loc2)$, that in STRIPS language is:

```
Action move(loc1, loc2):
  Preconditions: prec: Adj(loc1,loc2), AgentAt(loc1), ¬WumpusAt(loc2)
  Effect: ¬ AgentAt(loc1), AgentAt(loc2).
```

The observable propositions are $smell(x, y)$ indicating there is a smell in the position (x, y), which implies in the presence of the Wumpus in an adjacent cell. Notice in the solution depicted in Figure 3.13, that the initial belief state b_0 contains two (physical) states, and each of them differs only on the Wumpus position. When the agent performs an observation, for example at location $(2, 1)$ the *belief tracking* operation (Equation 3.8) obtains the successor belief states B_4 or B_5 according to the observation of the literal $smell(2, 1)$.

Since contingent planning problems consider observations and non-deterministic actions, the solution is no longer a sequence of actions. When planning with partial information and sensing, the **solution** for a contingent planning problem P is a contingent plan, or a *policy*, i.e. a partial function that maps belief states b into actions.

Definition 14 (POND policy)**.** *A (stationary) policy for* POND *planning problems (conformant or contingent) is a mapping from* ***belief states*** *to actions, i.e., $\pi : 2^S \rightarrow A$, prescribing the action $\pi(s)$ that must be taken in the state s. A* POND *policy can be one of the three types: weak, strong or strong cyclic. $\pi(b)$ maps b to action a if $prec(\pi(b))$ holds $\forall s \in b$.* ◇

Figure 3.14 shows examples of different plans for each type of solution. The POND policy can be seen as a graph, where each node represents a belief state and an edge represents the action prescribed by the policy, i.e., $\pi(b)$. A leaf node maps the belief state b into a *noop* action (an action with no effects and preconditions) indicating that the plan is finished.

3.2.3 Contingent problems with Dead-Ends

In contingent problems, since we work over a belief state space, we have to define *dead-end belief states* (DEB), which are belief states from which there is no POND policy (weak, strong or strong cyclic). Due to the uncertainty in the initial state, a *dead-end belief state* b can be of three types:

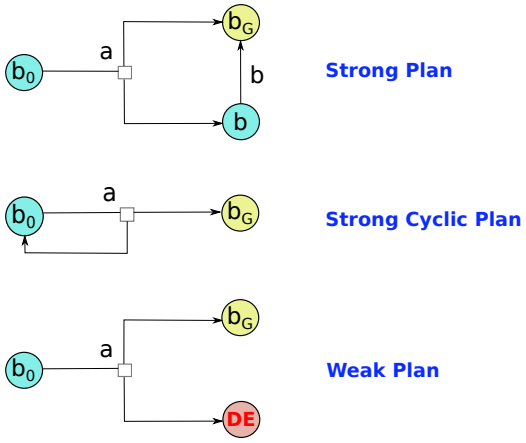


Figure 3.14: *Examples of different types of* POND *policies for contingent planning, where b_i are belief states, b_G is a belief goal state and DE is a dead-end belief state.*

- **DEB1**: b is a dead-end belief state of type DEB1, if $\forall s \in b$, s is a (physical) dead-end state.
- **DEB2**: b contains only *some* pure dead-end states, there exists a *weak plan solution*; b is a dead-end belief state of type DEB2 if $\exists s \in b$, such that s is a dead-end state.
- **DEB3**: b is a dead-end belief state of type DEB3 if $\nexists s \in b$, such that s is a dead-end but there is no action that can be applied in b that leads the agent to the goal, i.e. the uncertainty in b is such that the agent cannot come up with a strong or weak POND policy.

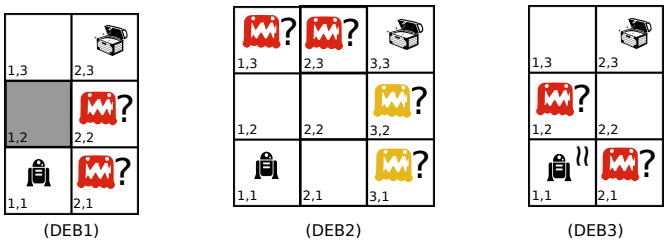


Figure 3.15: *Example of three Wumpus World instances with belief dead-end states; (left) Wumpus World 2x3 where there is a wall in the location (1,2) and a Wumpus either at (2,1) or (2,2); (middle) Wumpus World 3x3 with two different Wumpus, one at (1,3) or (2,3) and the other at (3,1) or (3,2); (right) Wumpus World 2x3 with a Wumpus either at (1,2) or (2,1).*

Figure 3.15 (left) shows an instance of Wumpus World problem that serves as example of dead-end belief state of type DEB1, where there can be two possible Wumpus locations, (2,2) and (2,1), and no solution can be found for each one of those possible states, since $\forall s \in b_0$, s is a dead-end state. Figure 3.15 (middle) shows another instance of Wumpus World problem with an example of DEB2, with 4 possible physical states but with only one being a dead-end, i.e., the state that satisfies $WumpusAt(2,3) \wedge WumpusAt(3,2)$; Figure 3.15 (right) shows an example of dead-end belief state of type DEB3, where the agent believes the Wumpus can be either at location (2,1) or (1,2), since these two locations are not safe no action is applicable. The two cases of DEB1 and DEB3 (left and right) will cause the agent to *freeze* while in the middle case the agent can come up with a solution.

Dead-end belief states of type 1 DEB1 are similar to dead-end states in the FOND setting, that is, there is no action applicable, or all actions lead to states that are also dead-ends, never reaching the goal. But dead-end belief states of type DEB2 and 3 DEB3 appears only in POND problems, due to the uncertainty in the initial state. Dead-end belief states of type DEB2 induce

weak policies, because as the agent perform observations it will minimize the size of the belief state, finding policies for all states that are not dead-ends. However, when the only state left in the belief state is a dead-end, there is no solution and the search stops, returning a weak policy.

It is possible that a contingent planning problems presents a combination of these dead-end belief states. For example, consider Figure 3.16 that shows an example of a contingent planning problem with dead-end belief states of type DEB3, caused exclusively due to the uncertainty in the initial state, for which there is no strong solution, that is, if the agent acts any action may lead to a dead-end. Even if there exists a strong policy from the physical states belief state b_0, the combination of these states makes that no action is applicable in both states, causing that every action applied in b_0 will lead the agent towards a dead-end belief state of type DEB1 or DEB2, as it can be seen in Figure 3.16.

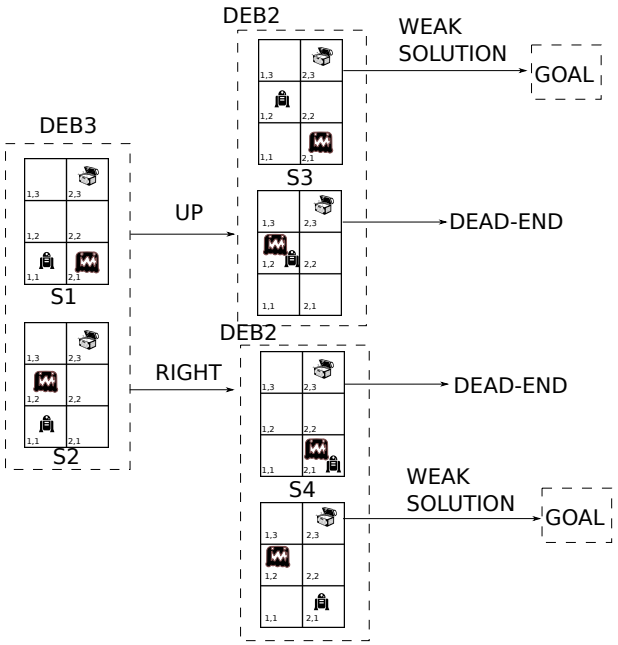


Figure 3.16: *Example of an initial belief state for the Wumpus problem, where the Wumpus can be either at location* $(1, 2)$ *or* $(2, 1)$*, causing the initial state to be a dead-end belief state of type 3.*

3.2.4 Solving a Conformant Planning Problem

The first work that explicitly considers a translation of the original conformant problem to a classical planning problem is given by Palacios and Geffner (2006). This translation can result in a loss of completeness, that is, not all solutions of the original problem solve the translated version, but it can have efficient solutions for the problems that it solves, due to the use of state-of-the-art classical planners. However, it is important to notice that these translations may only work when there are no dead-ends in the problem (Albore and Geffner, 2011; Bonet and Geffner, 2013, 2014b), and otherwise, they offer no guarantee to compute even a weak policy, if one exists. The translation approach, proposes to translate the conformant planning problem $\mathcal{M}$ into a deterministic planning problem $K(\mathcal{M})$, where the agent starts reasoning about the **truth knowledge** over the literals in $\mathcal{M}$, instead of reasoning about the world itself.

For a conformant problem $\mathcal{M}^{\text{CONF}}_{\text{STRIPS}} = \langle F, I, Op, G\rangle$ we define the translated problem $K_0(\mathcal{M}^{\text{CONF}}_{\text{STRIPS}}) = \langle F', I', Op', G'\rangle$ as follows:

Definition 15 (K_0-Translation)**.** *Given a conformant problem* $\mathcal{M}^{\text{CONF}}_{\text{STRIPS}} = \langle F, I, Op, G\rangle$*, the translated problem* $K_0(\mathcal{M}^{\text{CONF}}_{\text{STRIPS}})$ *is a tuple* $\langle F', I', Op', G'\rangle$ *where:*

- $F' = \{KL, K\neg L | L \in F\}$*;*

- $I' = \{KL, K\neg L | L \in I\} \cup \{\neg KL', \neg K\neg L' | L' \notin I\}$;
- $Op' = Op$, *but replacing every literal* L *in the precondition of the action* $a \in A$ *for* KL, *and each effect* $e \in eff(a)$, $e : \langle add(e), del(e)\rangle$ *is replaced by an effect* $e : \langle\ Kadd(e) \cup K\neg del(e), K\neg add(e) \cup Kdel(e)\rangle$; *and*
- $G' = \{KL | L \in G\}$.

◇

The meaning of the operator K is to indicate if the value of the variable L is known by the agent. As an example, KL means that the value of L is known to be true, and $K\neg L$ that L is known to be false. Similarly $\neg KL$ means that the agent does not know if L is true, and $\neg K\neg L$ that it does not know if L is false. $Klist$ stands for $Kc_1, Kc_2, ..., Kc_n\ \forall c_i \in list$, and similarly $\neg K\neg list$ stands for $\neg K\neg c_1, \neg K\neg c_2, ..., \neg K\neg c_n\ \forall c_i \in list$. Thus, all the uncertainty from the initial state of $\mathcal{M}^{\text{CONF}}_{\text{STRIPS}}$ is removed in $K_0(\mathcal{M}^{\text{CONF}}_{\text{STRIPS}})$. The K_0-Translation is based on epistemic logic (Fagin et al., 2003; Van Ditmarsch et al., 2007) (for a brief review of epistemic logic see Appendix B) and the axioms important to this work are:

Necessitation Axiom The **Necessitation** Axiom (**N**) states that valid formulas in a state are known by the agents, i.e., $\phi \rightarrow K\phi$. These are the formulas that are *necessarily* true, as opposed to the formulas that just happen to be true at a given state.

Truth Axiom The **Truth** Axiom (**T**) expresses that the knowledge of the agent is in fact true $K\phi \rightarrow \phi$. If the agent knows a fact ϕ, then this fact must be true: $K\phi$.

The **Necessitation Axiom N** of epistemic logic (Fagin et al., 2003; Van Ditmarsch et al., 2007) is straightforwardly mapped by the translation, as every literal L known to be true in the initial state is translated as KL, meaning that the value of L is known by the agent . This ensures correctness as prevents the planner to generate a state containing both KL and $K\neg L$.

Translation $K_0(\mathcal{M}^{\text{CONF}}_{\text{STRIPS}})$ is incomplete, but it has the following properties (Palacios and Geffner, 2006):

- Soundness: If π is a policy that solves the translated classical planning problem $K_0(\mathcal{M}^{\text{CONF}}_{\text{STRIPS}})$, then $K^{-1}(\pi)$ is a policy that solves $\mathcal{M}^{\text{CONF}}_{\text{STRIPS}}$, where $K^{-1}(\pi)$ is the inverse translation from the epistemic states to the original states.
- K-literals: If π is a policy that leads to KL, then π is a policy that leads to the literal L with certainty.

The second property ensures that if an agent reaches a state where KL holds, the sequence of actions that the agent applied to reach KL will also reach a state where L holds in the original problem, satisfying the **Truth Axiom T**.

The K_0-Translation is sound but not complete, because this basic translation it is not capable of reasoning with disjunctions, and when the initial situation includes some kind of disjunction, this translation it is unable to translate it, because it only translates what it is known (Palacios, 2009). For example, consider the following conformant planning problem P, with $F = \{p, q, r, v\}$, $I = \{p \vee q\}$, $G = \{q\}$ and action a with no preconditions and a single conditional effect $\langle p, \langle q, \emptyset\rangle\rangle$, that is, an effect that will be triggered only if p holds in a state. The conformant plan $\pi = \{a\}$ is valid since for both possible initial states it will end in state $\{q\}$. However, the $K_0(P)$ translates the initial state as $\{\}$, that is no literal is known, hence applying the translated action a will result in an state where Kq does not hold.

To achieve completeness, the translation $K_0(\mathcal{M})$ can be extended to the translation $K_{T,M}(\mathcal{M})$ (Palacios and Geffner, 2007b) that introduces the concept of *tags* and *merges*, but the translated problem increases its size exponentially. For an example of an incomplete translation, see Appendix C, and for more details of the $K_{T,M}(\mathcal{M})$ Translation, see Appendix D.

3.2.5 Solving a Contingent Planning Problem

So far, we have seen the translation approach to solve conformant planning problems. In this section we extend this translation to solve contingent planning problems.

The K_1-Translation (Bonet and Geffner, 2011) is an incomplete but sound translation, based in the incomplete translation K_0-Translation for conformant planning problems (Palacios and Geffner, 2009), extended to include observations.

For a contingent planning problem $\mathcal{M}^{\text{CONT}}_{\text{STRIPS}} = \langle F, I, A, O, G\rangle$, the $K_1(\mathcal{M}^{\text{CONT}}_{\text{STRIPS}})$ will generate a fully observable non-deterministic problem, defined as $K_1(\mathcal{M}^{\text{CONT}}_{\text{STRIPS}}) = \langle F', I', A', G'\rangle$, where:

Definition 16 (K_1-Translation).

- $F' = \{KL, K\neg L \mid L \in F\}$;
- $I' = \{KL \mid L \in I\}$;
- $G' = \{KL \mid L \in G\}$;
- A' *is the set of actions* $A_A \cup O_A$ *where:*
  1. A_A : *for every action* $a \in A$, *there is an action* $a_a \in A'_A$ *where every literal* $L \in prec(a)$, *is replaced by* KL, *and for every effect* $e \in eff(a)$, $e : \langle add(e), del(e)\rangle$ *of the action* a, a_a *returns a translated effect* $e : \langle Kadd(e) \cup K\neg del(e), K\neg add(e) \cup Kdel(e)\rangle$;
  2. O_A : *for every conditional observation* $o \in O$ *such that* $o = \langle prec(o), L\rangle$, *there is a non-deterministic action* $o_a \in O_A$ *such that every precondition* C *of* o, *is replaced by* $(KC, \neg KL, \neg K\neg L)$, *and* L *is replaced by two non-deterministic effects* $(KL \vee K\neg L)$;

◇

Notice that the translation $K_1(\mathcal{M}^{\text{CONT}}_{\text{STRIPS}})$ is similar to the incomplete original translation $K_0(\mathcal{M}^{\text{CONT}}_{\text{STRIPS}})$ (Definition 15).

As explained above, the solution to $K_1(\mathcal{M}^{\text{CONT}}_{\text{STRIPS}})$ is a policy π mapping the epistemic states to actions in A'. A strong (cyclic) policy π solves $K_1(\mathcal{M}^{\text{CONT}}_{\text{STRIPS}})$ if and only if every possible execution according to π reaches a epistemic state satisfying the goal G', i.e. a state where literals $KL \in G'$ are true, and a weak policy solves $K_1(\mathcal{M}^{\text{CONT}}_{\text{STRIPS}})$ iff some possible execution of π reaches an epistemic state satisfying the goal G'.

3.2.6 Translation-Based Planners

In this section we briefly describe two of the main planners that use these translations in order to solve contingent planning problems: CLG (Albore et al., 2009) and PO-PRP (Muise et al., 2014). Both planners work in two phases as seen in Figure 3.17 (in Appendix A we show other contingent planners that do not use translations like *Contingent*-FF and SDR).

Contingent Loop Greedy (CLG)

To solve the contingent planning problem M, the CLG planner makes a translation called $X(\mathcal{M})$ which includes *tags* and uses hypothetical reasoning over them. CLG planner also uses a relaxed version of the problem called the *Heuristic Model* $H(\mathcal{M})$, that is a deterministic version of the original planning problem, to obtain an estimate of the size of the solution size.

CLG uses this translated version $H(\mathcal{M})$ to select which are the best actions to expand. CLG works by computing an ordered sequence of actions σ until an observation is selected. When an observation is selected to be applied, the planner stops and recursively tries to solve every branch of the observations. The sequence of actions σ is obtained by the use of a classical planner, in this case using the FF-planner (Hoffmann and Nebel, 2001).

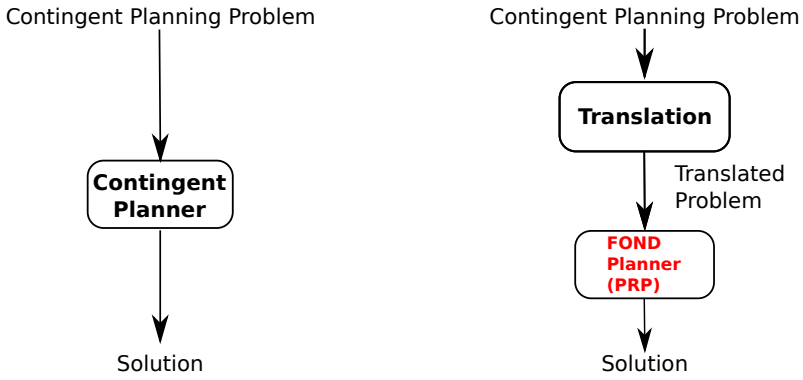


Figure 3.17: *Different architectures for contingent planners: (left) Contingent planners that solve the problem without translating it first; (right) Contingent planners that solve the problem by translating and using a* FOND *planner.*

Partially Observable Planner for Relevant Policies (PO-PRP)

The planner *Partially Observable Planner for Relevant Policies* (PO-PRP) (Muise et al., 2014) uses also a translation. In fact this planner uses the basic contingent translation showed in Definition 16. This translation is then used as input for the PRP planner (Muise et al., 2012), described in Section 3.1.3.

3.3 Discussion about this chapter

In this chapter we have presented the foundations of non-deterministic planning problems, with full or partial observability. We have also shown how to solve POND problems, using a state-of-the-art approach to solve these problems by translating them into problems with full observability in the epistemic state space. However, we have also showed how dead-ends present a major challenge for solvers, and how it is difficult to come up with solutions in the presence of dead-ends.

As we have mentioned, the K-translation can fail in the presence of dead-ends. Thus, since the objective of this thesis is to investigate how to use human help in POND problems with dead-ends, in the next chapter we show how the solutions based on incomplete K-translations for conformant or contingent planning problems with dead-ends can guarantee to find strong solutions when we introduce human help actions.

Chapter 4

Human Help in Contingent Planning

As we have stated in the previous chapter, we assume that an agent can only accomplish its task if it can find a strong (cyclic) plan, possibly including human help for observation and actuation when necessary. I.e., we are dealing with a situation for which weak solutions are not acceptable. An agent in any of the dead-end situations defined in Section 3.2.3 can pro-actively ask humans for help: in the DEB1 and DEB2 cases, a human could be asked to modify the environment to allow the agent to continue planning; in the DEB3 case a human can help to reduce the agent uncertainty by performing an observation the agent is not able to make.

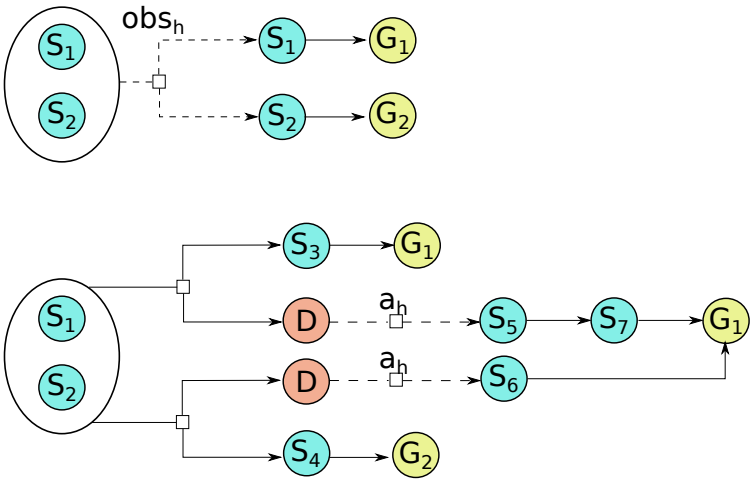


Figure 4.1: *Examples of* POND *belief state space with dead-ends of type DEB2 and DEB3 dead-ends. (top)* POND *with DEB3; the human observation o_h allows the agent to distinguish between the states s_1 and s_2 of the initial belief state; (bottom)* POND *with DEB2; the human actions a_h allows the human to modify facts of the world, so the agent can continue planning to the goal.*

Thus, human help can be used: (i) to modify facts of the environment when the agent cannot find a solution from the current state, or (ii) to observe facts that are unknown to the agent that is unable to make such observation. As an example, consider the contingent planning problem with dead-ends of type DEB2 and DEB3 shown in Figure 4.1. In this example, when the agent encounters a dead-end DEB3 (Figure 4.1 (top)), it is unable to continue planning because no action can be applied. But with the help of the human observations obs_H are considered, the agent is able to distinguish the states of the belief state and find a strong policy. On the other hand, when the agent encounters a dead-end of type DEB2 (Figure 4.1 (bottom)), the agent is unable to reach the goal after performing an action, because each action has an outcome leading to a dead-end but with the help of the human action a_H, the agent can also find a strong policy.

4.1 Human Help Observations

Consider a search and rescue mission robot in an environment with partial observation. To rescue a victim the robot must reach a location with an unknown safe access. In this situation, the robot can ask a human from a different location (e.g. from an helicopter), to perform an observation for a possible safe access to the victims location. With this new information, the robot may be able to accomplish its mission, otherwise it would fail.

Initially, we assume the human is capable of observing the truth value of any observable fluent of the problem $\mathcal{M}^{\text{CONT}}_{\text{STRIPS}}$, thus helping the agent to disambiguate states in a belief state.

Given a contingent planning problem $\mathcal{M}^{\text{CONT}}_{\text{STRIPS}}$ with a set of conditional observations O, where each $o \in O$ is of the form $\langle prec(o), l\rangle$ (Definition 13), the set of human observations O_H is constructed over the set O of $\mathcal{M}^{\text{CONT}}_{\text{STRIPS}}$ but with a relaxation of its preconditions.

Furthermore, as the K_1-Translation suggests we must decouple the observations that are triggered by the agent action, so they can be performed by the human. So, the new contingent planning problem with human help $\mathcal{M}_H$ still has the agent's observation being triggered by its actions but includes a set of human observation actions O_H, where each $obs_H \in O_H$ of an observable proposition L is a human observation of the form:

Definition 17 (Human Observations)**.** *The set of human observations O_H for an $\mathcal{M}^{\text{CONT}}$ with the set of conditional observations O, is defined for each $o \in O$, where $o = \langle prec(o), l\rangle$ as:*

```
Action obs_l:
  Precond: ∅
  Effect: l ∨¬l
```

◇

From a FOND planner point of view, a human observation is a regular action, and its K_1-Translation is:

```
Action obs_H(l):
  Precond: ∅
  Effect: (Kl ∧ ¬ K¬l) ∨ (K¬l ∧ ¬ Kl)
```

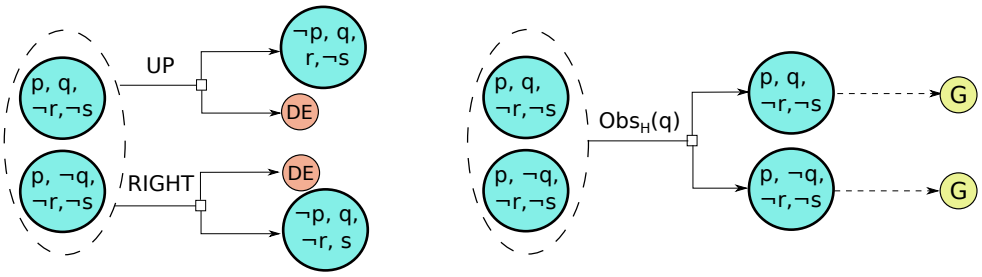


Figure 4.2: *Example of belief space of the Wumpus World problem shown in Figure 3.16, with propositional encoding p, q, r, s; (left) without human help there is only a weak policy; (right) with the human observation help there is a strong policy.*

To illustrate how human observations allow the agent to overcome dead-end belief states of type DEB3, consider the Wumpus World problem with a dead-end of Figure 3.16. Suppose that the proposition q stands for `WumpusAt(1,2)`, and $\neg q$ for $\neg$`WumpusAt(1,2)`. Similarly p, r, s stand for the position of the agent `at(1,1)`, `at(2,1)` and `at(1,2)` respectively. The original problem, using this propositional encoding is shown in Figure 4.2 (left). After applying any action, there is a possibility that the agent ends up in a dead-end, that is, at most there is weak solution as shown in the outcomes of the actions in Figure 4.2 (left). But if the agent resorts to the human observations $obs_H(q) = smells(1,3)$, the human will eliminate the agent's uncertainty who will then be able to come up with a strong policy (Figure 4.2 (right)).

Theorem 1. *Let $\mathcal{M}^{\text{CONT}}$ be a contingent planning problem with a dead-end belief state of type DEB3. If the translated problem $K_1(\mathcal{M})$ is augmented with the set of human observations O_H, a* FOND *planner is able to find a strong (cyclic) policy for it.*

Proof. The intuition of this proof is simple. Since the human can observe the truth value of any observable proposition, when the agent is unable to continue planning it may ask for the human to perform an observation. This helping action could be performed in any belief state, because the observation has no preconditions, dividing the belief state in two, with those states in which the observed literal holds in one half, and those states where the observed literal does not hold in the other. The human could perform this observations recursively in each resultant belief state until there is no more uncertainty and the size of the belief state is one. Since, by definition, DEB3 have no real (physical) dead-end states, there exists a strong policy from each of those real states. □

We must also note that the incompleteness of the K_1 translation do not affect the validity of Theorem 1, because using human observations, the agent is able to end with all uncertainty, that is the cause of the incompleteness (Palacios and Geffner, 2006, 2007a).

We could also restrict the set of human observations, by analyzing the causal graph of the determinized version of $\mathcal{M}^{\text{CONT}}_{\text{STRIPS}}$.

4.2 Human Help Actions

Dead-end belief states of type DEB1 and DEB2 (Figure 3.15 (left) and (middle)) cannot be solved neither by the agent, nor by human observations alone. However, the agent can try to ask a human to modify the environment. Based on the work of Göbelbecker et al. (2010), given a $K_1(\mathcal{M})$ translation of a contingent planning problem $\mathcal{M}$, and an epistemic dead-end state s_{DE} we generate the closest epistemic state of s_{DE} from which there is a strong policy. The idea is to create new actions named *human actions*, that can change the truth value of a literal.

Thus, the set of human actions A_H defines an action $a_H(l) \in A_H$ that changes the literal l to $\neg l$.

Definition 18 (Human Actions)**.** *The set of human actions A_H for an $\mathcal{M}^{\text{CONT}}$ with propositions F is defined for each literal l over F as:*

```
Action a_H(l):
  Precond: ¬l
  Effect: l
```

◇

From a FOND planner point of view, a human action is a regular action, and its K_1-translation is:

```
Action a_H(l):
  Precond: K¬l ∧¬Kl
  Effect: Kl ∧¬K¬l
```

Notice that we force the agent to verify the truth value of the literal it needs to change, to avoid the agent changing observable literals.

Theorem 2. *Let $\mathcal{M}^{\text{CONT}}$ be a contingent planning problem with dead-end belief states of types DEB1, DEB2 and DEB3. If $\mathcal{M}^{\text{CONT}}$ is augmented with human actions and observations (A_H and O_H), then a* FOND *planner is able to find a strong (cyclic) solution for the translated* FOND *problem $K_1(\mathcal{M}^{\text{CONT}})$.*

Proof. Since human actions have no preconditions and can modify the value of any literal of the $K_1(\mathcal{M}^{\text{CONT}})$, any goal state can be reached from any state. □

4.3 Human Help Contingent Planning Problem

An agent in any of the three dead-end belief state situations can proactively ask humans for help: in **DEB1** and **DEB2** cases, a human could be asked to modify the environment so the agent can continue to plan; in **DEB3** case a human can help to reduce the agent uncertainty by performing an observation the agent can not do.

We define then a new class of problems that is able to use human help to overcome the dead-ends.

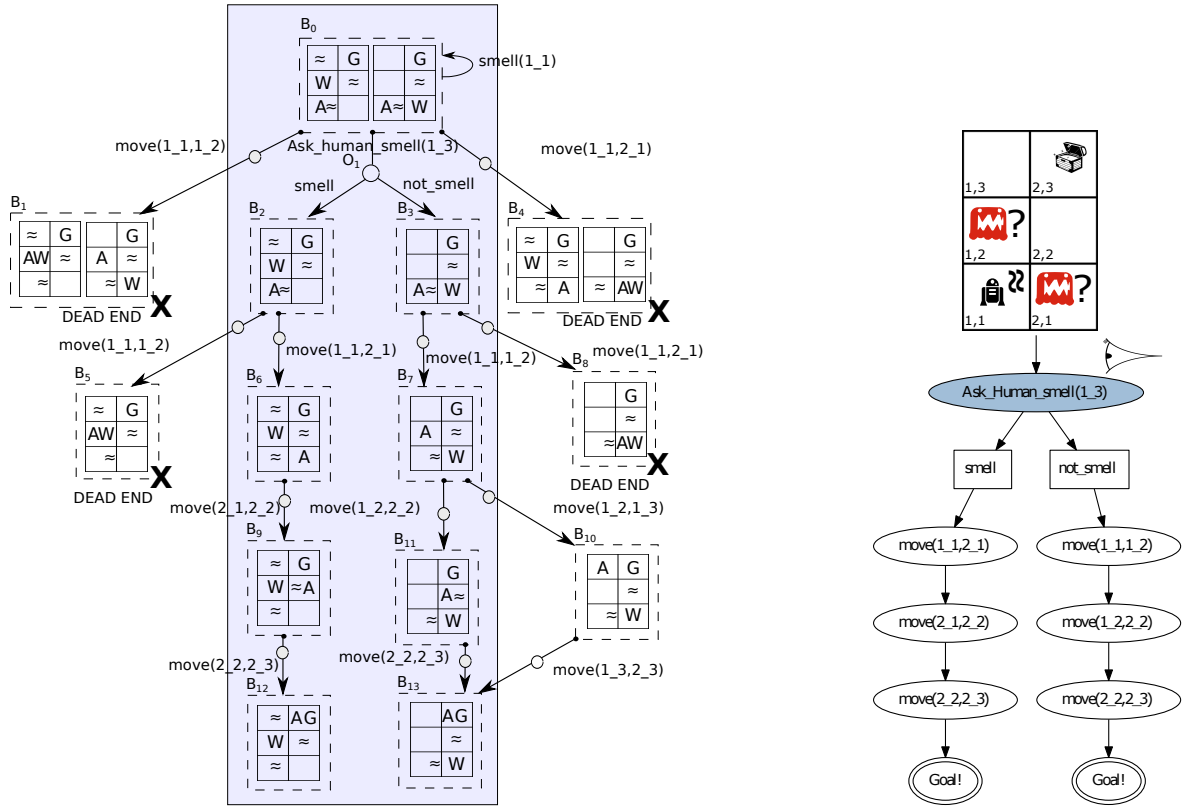


Figure 4.3: *(left) The* AND/OR *belief space with a solution (grey region) that includes a human observation; (right) contingent plan extracted from the search that branches on the human help observation at location (1,3).*

Definition 19 (Human Help Contingent Planning Problem)**.** *Let* $\mathcal{M}^{\text{CONT}}_{\text{STRIPS}} = \langle F, I, A, O, G\rangle$ *be a contingent planning problem, an Human Help Contingent Planning Problem (*HH-CP*) is a tuple* $\mathcal{M}^{\text{HHCP}}_{\text{STRIPS}} = \langle F, I, A', O', G\rangle$ *where* F*,* I *and* G *are defined as in* $\mathcal{M}^{\text{CONT}}_{\text{STRIPS}}$*, and:*

- $A' = A \cup A_H$*, where* A *is the set of the agent actions and* A_H *is the set of human actions, where* $a_H(l) \in A_H$ *is as in Definition 18,*
- $O' = O \cup O_H$ *is the set of the agent observations and* O_H *is the set of human observations, where* $obs_H(l) \in O_H$ *is as in Definition 17,*

◇

As we have mentioned before, a FOND planner (e.g. PRP) can find policies that consider non uniform cost. Thus, to solve a $K_1(\mathcal{M}^{\text{HHCP}}_{\text{STRIPS}})$ that uses human actions and observations only when necessary, we define the cost of human actions and observations higher than the cost of agent actions. Moreover, we assume that the human would prefer to help with an observation than with an action, so we also define a higher cost for the human actions when compared to human observations, i.e.:

- $C_a : A' \to \mathbb{R}^+$ is the cost function that defines cost=1 to the agent actions and cost > 1 for the human actions, and
- $C_o : O' \to \mathbb{R}^+$ is the cost function that defines cost=1 to the agent observations and cost > 1 for the human observations.

Figure 4.3 (left) shows the belief space search for an instance of the Wumpus World domain (Figure 3.16 (right)). Notice that all the expanded states without human help are dead-ends (Figure 4.3 (left)). Figure 4.3 (right) shows a strong plan with a human help observation at the root.

Theorem 3. *Given a human help contingent planning problem $\mathcal{M}^{\text{HHCP}}_{\text{STRIPS}}$, any* **FOND *planner*** *solves K_1-translation, i.e., $K_1(\mathcal{M}^{hhcp}_{\text{STRIPS}})$ using human actions and observations only when a dead-end is reached.*

Proof. The proof is quite straightforward. By Theorems 1 and 2 we know that it is always possible to find a strong policy if the agent resort to human actions and observations, and pays a higher cost. This cost will prevent the agent to select these actions if there is a path to the goal without using human actions. But with problems where there exist only weak solutions, any optimal FOND planner is able to recognize dead-end states during the search, either because the state has no applicable actions, or because it is part of a cycle and never reaches the goal, and it will assign a higher value to this state. The cost of a dead-end is always higher than those of these human actions, the agent will then ask for human help. □

4.4 Relevant Human Actions

In the previous section, given a contingent planning problem $\mathcal{M}$, we assumed that the set of human help actions A_H was defined over all propositions of $\mathcal{M}$, i.e., $\forall f \in F$. Even if this sets guarantees that a strong policy exists, it is highly inefficient to solve a planning problem where the agent is able to ask the human to change the value of any literal of the problem, due to the high branching factor. However, it is possible to restrict this set of literals to a much smaller set of literals without losing this guaranty. This set of literals $F_r \subset F$, called the set of **relevant literals** of $\mathcal{M}$ can be used to define a smaller set of human actions A_H without losing the guarantee to find a strong policy.

In this section, we start by describing an STRIPS language adaptation of the *Domain Transition Graph* and the *Causal Graph* (Helmert, 2006), first introduced in Section 2.1.6, enconding all the dependencies between the different variables of the problem. Next, we show some properties from these graphs like how to obtain set of relevant literals to create the reduced set of human actions A_H, as well as how to detect some kind of dead-ends from the information present on these graphs.

4.4.1 Boolean Causal Graph

The *domain transition graph* and the *causal graph* above have been defined for *multi-valued* planning task. However, in this work we use a STRIPS language with propositions instead. While transforming a *multi-valued* planning task to STRIPS language is quite straightforward, because it suffices to transform every assignment of the type $v = d$ into a proposition, the opposite is not as evident. In fact, multi-valued planning task encodes knowledge about values that cannot be true at the same time, for example if $v = d$ for a value $d \in D_v$, then it cannot be possible at the same time $v = d'$ for a value $d' \in D_v$ s.t. $d \neq d'$. But in a Boolean setting, it is necessary to encode this changes into the effects to avoid these situations. For example, when $v = d'$ is in a effect $\neg(v = d), \forall d \in D_v \setminus d'$ should appear in the same effect. Another solution could be to have *axioms* to prevent situations where two propositions $v = d$ and $v = d'$ could be true at the same time.

Since boolean variables can have only two values $\{\top, \bot\}$, that is, the two literals of a fluent, the domain transition graph of a variable is given by Definition 20.

Definition 20 (STRIPS Domain Transition Graph)**.** *The* STRIPS *domain transition graph (*DTG*) of a proposition p, denoted by* DTG*(p) is a labeled directed graph (V, E) with the set of nodes $V = \{\top, \bot\}$, that is a literal p and another literal representing the negation of the proposition $\neg p$. There exists an edge between $\top$ and $\bot$ if:*

- *There exists an action a in which p appears in the preconditions and in the $del(e)$ list of some effect e, in that case the edge is labeled with $prec(a) \setminus p$.*
- *$p \notin pre$, but p appears in the del list of some effect. In this case, the edge is labeled as $prec(a)$.*

◇

For the edges from $\bot$ to $\top$, it suffices to exchange p with $\neg p$.
And the causal graph is given by Definition 21.

Definition 21 (STRIPS Boolean Causal Graph)**.** *The* STRIPS *causal graph (*CG*) of a planning problem $\mathcal{M}$, denoted by* CG*$(\mathcal{M})$ is a labeled directed graph (V, E) where $V = F$, and there exists an edge $(x, y) \in E$ for $x, y \in V$ iff $x \neq y$ and there is an action $a \in A$ with an effect $eff(a)$ such that $y \in add(e)$ or $y \in del(e)$, and $x \in Prec(a)$.* ◇

It is interesting to remark that while our work is focused on FOND problems (and also MDP as we will see later) and contingent problems, it is easy to perform the following adaptation to use the Causal Graph:

- Non-deterministic planning problems are *determinized*, that is, for each non-deterministic effect e_i of a non-deterministic action a, we create a different *deterministic* action a_i.
- Observations are transformed into two actions: one with the effect of the fluent observed being true and other with the fluent observed being false.

4.4.2 Relevant Literals

An interesting property of the *Causal Graph*, is that it allows to obtain information about which literals are relevant to the goal, that is, which literals participate in actions leading to the goal state. A fixed point iteration to obtain the set of *relevant* literals to the goal $Rel(G)$ of a planning problem $\mathcal{M}^{\text{CONT}}_{\text{STRIPS}}$ is shown below:

- Initialize $Rel(G)$ with the literals appearing in the goal;
- If there exists an edge in the CG between a literal p and another literal $q \in Rel(G)$, then p is added to the set $Rel(G)$;

The procedure above is repeated until a fixed point is reached and no more literals are added to the set $Rel(G)$. These literals are assumed to change their value, except for literals already in the goal state, so we call this set the *Relevant literals*. If any literal that was marked as relevant because it was present in a precondition do not change its value, it would be treated as an invariant and deleted from the problem (Edelkamp and Helmert, 2000).
We could use this set of relevant literals to create a set of human actions A_H as we show next.

4.4.3 Human Help Actions from Relevant Literals

In Section 4.2, given a contingent planning problem $\mathcal{M}^{\text{CONT}}_{\text{STRIPS}}$, the set of human help actions A_H was defined over all lierals of $\mathcal{M}^{\text{CONT}}_{\text{STRIPS}}$, i.e., $\forall f \in F$. However, we can find optimal solutions with the set of human help actions A_H defined only over a subset of propositions $F_r \subset F$, called the set of **relevant literals** of P, without losing the guarantee of always being possible to find a strong policy.

Theorem 4. *Given a contingent planning problem $\mathcal{M}$ and its corresponding* HH-CP *$\mathcal{M}^{hhcp}$, the human help actions in the strong policy for $\mathcal{M}^{hhcp}$, are actions that change literals from the set of relevant literals (F_r) of the determinized version of P.*

Proof. Let us consider a HH-CP $\mathcal{M}^{hhcp}$ problem for which there is no strong policy without human help, and a human help action a_w that changes the literal $w \notin F_r$. Let us suppose that a_w is the optimal action for state s. Then, if a_w does not help achieving the goal, an extra human help will be necessary, which will increase the size of the strong policy π^*. So, the optimal action for s should be an action that changes a relevant fluent, i.e., a fluent $f \in F_r$, which contradicts the former assumption of a_w is the optimal action for state s. □

4.4.4 HH-CP planner

The planner HH-CP planner (*Human Help Contingent Planning*) (Andrés et al., 2017; Franch, 2017) is based on the computation of the set of relevant literals. This planner works on two phases: (1) the planner gets as input a contingent planning problem and creates a set of human actions and observations from the relevant set of literals of the problem $\mathcal{M}^{\text{CONT}}_{\text{STRIPS}}$, producing a translated version of the problem $\mathcal{M}_h$; and (2) a layer with a FOND planner to solve the translated problem (a modified version of the PRP planner Muise et al. (2012)). This planner can be set to use: (1) only human observations; (2) only human actions; or (3) first produce a solution using only human observations and then correcting the solution by simulating the plan and using human actions.

The set of actions A_H created from the set of relevant literals is still a large set. In some cases it may present *trivial* human actions, i.e. human actions that make true a fluent on the goal. These actions, even if formed from relevant literals, are not descriptive of the failure of the agent to reach the goal. In the next section we introduce the concept of excuse, and how we can find a smaller and most descriptive set of human actions A_H, that are not trivial.

4.4.5 Reducing the set of Human Actions

The relevant set is indeed an interesting set of literals with some properties. However, as shown in previous sections, this is still a large set, and the set of human actions A_H created from it may contain some trivial actions, that is actions that make the literals in the goal true directly. For example, consider a robot in a grid, where it can move in each direction but there is a probability that it can break the wheel. If we consider the set of human actions A_H formed from the set of relevant literals, when the robot breaks the wheel, regardless of its location, it is possible that the robot asks the human to take it directly to the goal location.

To avoid such cases, we want to restrict the set of human actions to only contain those actions that the robot is not able to do, that is, to get rid of trivial actions.

From the set of relevant literals $Rel(G)$, we only consider those literals for which there is no path in the *domain transition* graph, that is, *static literals* (Göbelbecker et al., 2010; Helmert, 2006). And we create only the human actions that modify the value of any of these literals. Thus, creating a human action a_H from a *static literal* represents a change that the agent is not able to do in the environment.

Göbelbecker et al. (2010) proved that the minimal change from a state that does not reach the goal, to the closest state from which there exists a solution will only contain *static changes*, i.e., changes to these set of literals.

As an example of an static change, let us consider the Doors problem depicted in Figure 4.4. Consider the following modification to the domain, where the key is located in an inaccessible position, that is behind the locked door at $(3, 2)$. In this problem, the set of relevant literals comprises all the locations, as well as the open door and the possible locations of the keys. However, as explained before, when the robot fails to get the key, a trivial change would be to get the robot to the goal directly. However, a better approach would be to ask the human for the key that unlocks the location $(3, 2)$, and go there directly. Consider now the Figure 4.5, and notice that there is no path in the *domain transition* graph for the literals that encode the position of the key, (x, y) from

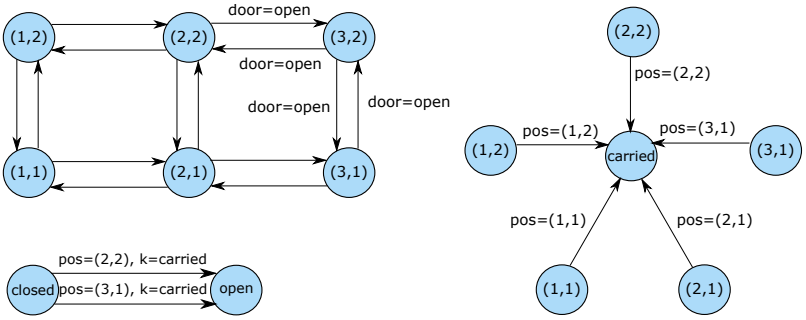


Figure 4.4: *Domain Transition Graphs (*DTG*) of the three variables of the Doors Problem. The first variable pos (top left) represents the position of the robot; the second variable door (bottom left) represents the state of the door; and the third variable k (right) represent the position of the key. Notice that once the key is carried it cannot be left again.*

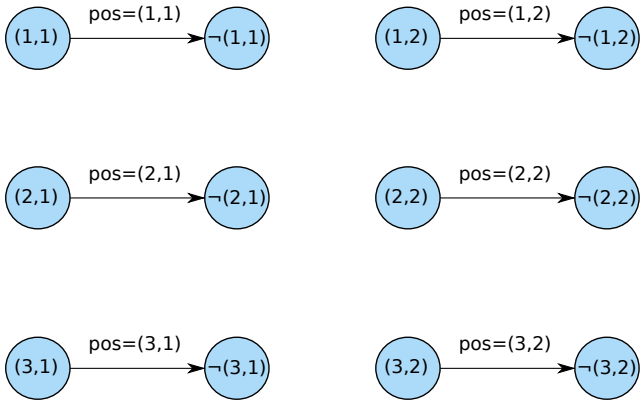


Figure 4.5: *Domain transition graph (*DTG*) of the Doors problem for a subset of Boolean variables, used to encode the position of the key. Notice that in Boolean setting, each Boolean variable has only two values.*

the literal $\neg(x, y)$ to the (x, y) literal. If we allow the human to perform the modification from a literal $\neg(x, y)$ to the literal (x, y) for any pair of literals encoding the position of the key, this is a static change, hence a *good human action*.

A static literal can also be seen as a particular case of a *one way fluent* (Edelkamp and Helmert, 2000). That is, fluents that only change value in one direction, that is, in a problem $\mathcal{M}$ they appear only in a *add* list or in a *del* list but never appear in both. Informally a *one way fluent* changes its value from *true* to *false* or *true* to *false* but not both.

4.4.6 Compact Compilation Belief Tracking Planner (COMP2BT)

The planner COMP2BT (*Compact Compilation Belief Tracking*) (Andrés and de Barros, 2016), is a planner that produces a new translation, able to solve contingent planning problems with dead-ends, by detecting potential dead-ends, and enhancing the translation with safer actions. To do so, it computes the set of static literals, detecting the set of actions that may add this literals. The K_1-Translation then includes a new precondition to prevent these actions to be selected if it might result in a dead-end, and axioms that help to reduce the belief state after a new observation is applied.

Definition 22 (Dead-end effect)**.** *(Andrés and de Barros, 2016) If action a has an effect e, such that a literal L is make true after applying e and the following conditions hold: (1) the complement of L, $\neg L$ is a relevant fluent of $\mathcal{M}^{\text{CONT}}_{strips}$; and (2) in the Domain Transition Graph of the variable L there are no outgoing edges from L, then the effect e is called a* Dead-end effect *because it can lead to a dead-end belief state.* ◇

Thus, COMP2BT translates a $\mathcal{M}^{\text{CONT}}_{strips}$ problem with the K_1 translation detecting if any action contains a dead-end effect:

Algorithm 1 *Compact Compilation Belief Tracking* (Andrés and de Barros, 2016)

INPUT: Action a
OUTPUT: Translated action a

```
1: function VERIFY-ACTIONS-DEAD-ENDS(a)
2:     for effect e in a do
3:         if isDead-End-Effect(e, a) then
4:             a = RemoveEffect(e, a)
5:             axiom ax = AddAxiom(e, a)
6:         end if
7:     end for
8:     return Translate(a)
9: end function
```

The VERIFY-ACTIONS-DEAD-ENDS function verifies for each action if it contains a dead-end effect according to Definition 22 (line 3). If action a contains a dead-end effect e, this effect is removed from the action and added as an axiom (lines 4 and 5). Removing the effect involves adding preconditions to avoid selecting a to apply when there is risk of entering a dead-end. And adding e as an axiom means that applying action a will still trigger effect e (For more information about this planner please see Andrés and de Barros (2016)).

This planner is based in a two-layer architecture: a layer with the translation, and a layer with the FOND planner that solves the translated full observable planning problem. The aim of COMP2BT is to first make the translation (*compile*) and then reduce the belief state when calculating the next state using the deductive axioms, i.e., the set of axioms $D \subset I$, translated to actions without cost (*compact*). These deductive axioms enhance a basic translation and make it able to perform belief tracking in complex contingent planning problems. The translation layer receives as an input a contingent planning problem P and returns the translated full observable version of the problem, with some extra additions to the basic $K'(P)$ translations (Sections 4.4.5).

The underlying planner used in the experiments is a modified FF planner (Hoffmann and Nebel, 2001), with some differences respect to the original. First, the planner branches and recursively calls itself after every observation considering both outcomes of the observation to construct the solution. The second modification is that after selecting an action for expansion, the state s obtained is closed under the deductive axioms, meaning that the agent can reduce the belief state obtaining knowledge, thanks to the added effects of the actions and the observations. Remember that the deductive actions are axioms that can be selected by the planner and expanded with no cost after executing an action, to reflect the process of deducting new facts in belief tracking. This is the most critical difference since part of the effects is translated as axioms that have to be maintained all the time during the execution.

4.4.7 Compact Compilation Belief Tracking Planner with Human Help (COMP2BT+HH)

Based on the COMP2BT planner, we extend it to include human actions. The idea behind this extension is to detect the static literals as COMP2BT does, but instead of producing a safer action, it creates a new set of human actions with these literals as explained in Section 4.4.5, producing a translation that includes a set of tailored human actions for each problem.

Similar to COMP2BT, COMP2BT+HH is based in a two-layer architecture: one with the translation, and one with the FOND planner. However COMP2BT+HH differs on the underlying planner: COMP2BT uses a modified FF planner, and COMP2BT+H uses a PRP planner.

4.5 Discussion

In this section we have explained how an agent with human help actions is always able to find a strong solution. We have also shown how to create the set of human actions automatically from the description of the problem encoded in the domain transition and causal graphs, by obtaining the set of relevant literals. Furthermore, we also show how to reduce this set of literals, to create even smaller and more informative sets of human actions. In the experiments chapter, we will look at how using our ideas, contingent planners are able to compute strong (cyclic) policies to solve problems that have no strong solution.

Chapter 5

Experiments in Contingent Planning Problems with Human Help

In this chapter we evaluate our proposed solutions COMP2BT, HH-CP and COMP2BT+HH, for contingent planning problems with dead-ends and human help. We divided this chapter in three parts. First, we describe the different planning domains used to test our algorithms, explaining their characteristics and why they are interesting for this evaluation. In the second part we present the results of our planner COMP2BT to detect dead-end effects using the causal graph, and translating the original contingent problem into a FOND planning problem with safer actions, and compare it to other planners considered state-of-the-art. In the third part, we show how solve contingent planning problems with different types of dead-ends by using human help actions and observations, using our planner HH-CP and showing how it is able to obtain strong solutions, and finally we compare its performance to our planner COMP2BT+HH.

5.1 Contingent Planning Domains

The contingent planning problems used in this chapter were extracted from a set of planning domains (Albore et al., 2009): Color Balls, Wumpus World, Secret Doors, Localize and Contingent Logistics. They are all considered simple contingent planning problems (Section 3.2.2), except for the Localize domain.

5.1.1 Color Balls Domain

In this domain, a robot located in a grid world surrounded by walls, must collect a set of balls of different colors (red, blue, purple and green) and throw each one in the corresponding garbage. The robot is able to move to any direction (N, W,S and E, if there is no wall in this direction, pick the ball (*grab*) (if the robot is at the same position of the ball) and, once the robot is sure of the color, throw the ball (throw) to the correct garbage, once the robot is sure of its color. The robot is also able to perform 2 types of observations in any location: (1) observe if there is a ball (*sense-ball*); and once the robot is holding the ball (2) observe its color (*sense-color*).

The uncertainty in this domain is caused by the initial location of the balls and their colors, which is unknown. The balls can be located in any cell of the grid except the four corners, where the garbage cans are located, and it is possible to have more than one ball in the same cell. An instance in the Color Balls domain is characterized by the size of the grid, and the total number of balls. For example the instance *cballs4-1* is a grid of size 4×4 with one ball.

This domain contains no dead-ends. However, it has a large number of possible states, and a large branching factor due to the observations, causing this problem to be hard to solve.

Table 5.1 shows the statistics for 5 instances of Color Balls domain, in terms of size of the (physical) state space ($|S|$), the size of the belief state space ($|B|$), the size of the initial belief state ($|b_0|$) and the size of the set of actions ($|A|$). For example, the smaller instance (*cballs4-1*)

Instance	$\|S\|$	$\|B\|$	$\|b_0\|$	$\|A\|$
cballs4-1	784	2^{784}	48	544
cballs4-2	37632	2^{37632}	2304	832
cballs4-3	1806336	$2^{1806336}$	110592	1120
cballs10-1	38500	2^{38500}	384	11800
cballs10-2	14784000	$2^{14784000}$	147456	13600

Table 5.1: *Color Balls instances; cballsn_m corresponds to a grid $n \times n$ with m balls.*

has 48 possible initial states, $(12 \times 4 \times 1)$, that is the possible locations where the ball is situated (except the four corners) times the possible colors of the ball. The robot starts in a known location. The agent must compute a solution that can consider any of the 48 possible initial states. Let us remember that a belief state is a set of states deemed possible, and the total number of belief states is calculated as $2^{|S|}$. Also notice how the size of actions grows in a constant way. Every time an observation is applied, it divides the belief state in two.

5.1.2 Wumpus World Domain

Wumpus World is a contingent planning domain where an agent moves in a grid world, to find a treasure, and has to avoid locations with pits and monsters called *wumpus*. The agent does not know the position of the monsters or pits, but is able to sense if there is a danger in an adjacent location. The agent can enter in a cell with a *wumpus*, or falls in a pit, dying in either case. The agent is able to move in the grid in any direction (N,W,S and E), pick the treasure (*grab-gold*) and leave the grid. The agent is able to sense the smell of the wumpus (*smell*) and the breeze of a pit (*feel-breeze*) only if it is in an adjacent location of a pit or wumpus, respectively. Different from the original problem, the agent knows what are the possible locations of the monsters and pits. However, in this modified version the agent knows where the gold is. E.g., Figure 5.1 shows an instance of the Wumpus World problem where the initial location of the Wumpus is partially unknown, but the agent knows where the treasure is located, as well as its own location.

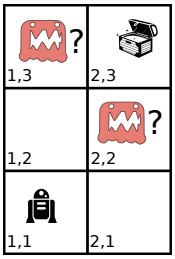


Figure 5.1: *Example of an instance of Wumpus World problem. The agent starts at location* $(1, 1)$*, and the treasure is located at* $(2, 3)$*. There is a wumpus either at location* $(1, 3)$ *or* $(2, 2)$*.*

In this version the agent must ensure the cell is safe before entering, by sensing and reasoning. This causes the domain to have no dead-ends, because the modeling of the problem prevents the agent from executing an action leading to a dead-end.

Thus, we also considered a different modeling of this problem, called *Wumpus-D*, where the agent moves to a location without verifying first if it is safe, possibly causing the agent to die, creating several dead-ends. This modified version appears in the benchmark domains of the MPSR Planner Brafman and Shani (2012).

Table 5.2 shows the statistics for the *Wumpus World* domains considered. Notice that the domain size remains the same between the two variations. In fact, the only difference is that there is a conditional effect for every movement action. This conditional effect states that when there is a *wumpus* or a pit in the destination cell, the agent will die. Notice that there is a *dead-end* for each possible initial position of the hazards.

Instance	$\|S\|$	$\|B\|$	$\|b_0\|$	Dead-ends	Actions
wumpus-5-3	400	2^{400}	16	0	700
wumpus-7-5	3136	2^{3136}	64	0	2548
wumpus-10-8	51200	2^{51200}	512	0	10300
wumpus-15-13	3686400	$2^{3686400}$	16384	0	51300
wumpus-d5-3	400	2^{400}	16	16	700
wumpus-d7-5	3136	2^{3136}	64	64	2548
wumpus-d10-8	51200	2^{51200}	512	512	10300

Table 5.2: *Statistics for the 7 instances of Wumpus World; wumpus$n-m$ corresponds to a grid $n \times n$ with m hazards.*

Instance	$\|S\|$	$\|B\|$	$\|b_0\|$	Dead-ends	Actions
secret-doors-7-3	16807	2^{16807}	343	0	2401
secret-doors-9-4	531441	2^{531441}	6561	0	6597
secret-doors-11-5	19487171	$2^{19487171}$	161051	0	14696
secret-doors-d5-2	625	2^{625}	25	16	35
secret-doors-d7-3	16807	2^{16807}	343	216	2401
secret-doors-d9-4	531441	2^{531441}	6561	4096	6597
secret-doors-d11-5	19487171	$2^{19487171}$	161051	10000	14696

Table 5.3: *Statistics for the 7 instances of Secret Doors; secret-doors$n-m$ corresponds to a grid $n \times n$ with m doors.*

5.1.3 Secret Doors Domain

The *Secret Doors* domain is a domain appearing in the benchmark domains of the CLG planner. However, its name has been changed here to avoid confusion with the Doors domain, used in other experiments, which is a completely different domain. In this domain a robot must move between rooms, finding the secret doors, and opening them to cross to another room. The goal of the agent is to reach the goal room.

The actions of the robot consist in moving in any of the four directions (N,W,S and E) to an adjacent cell and sense if there is a hidden door. When the agent finds the hidden door, it is able to open the door and cross. A variation of this problem *Secret Doors-D* removes the precondition of finding the door before crossing to a different room, and instead, adds an effect breaking the robot if it tries to enter a room through a wall.

The statistics for this problem are showed in Table 5.3. Notice how the number of *dead-ends* is bigger than in the *Wumpus World* problem. This is because, unlike in the *Wumpus World* problem in this domain the agent must find where the door is before crossing and *all* other locations are unsafe. In the *Wumpus World* the agent must avoid the location with the *wumpus*.

5.1.4 Localize Domain

This problem is also part of the benchmark domains of the CLG planner. Localize consist in a robot that must locate itself in a known map, and find its way to the goal location. However, the initial position of the robot is unknown.

This problem is encoded in such a way that it consists only in 4 actions, to move in the four directions, and 4 sensing actions. Each action, moves the agent in a cardinal direction (*north*, *west*, *south* and *east*) and has a conditional effect for each location, indicating the next location the agent will be located. The four sensing actions allow the agent to sense if there is a wall above him, to the left, right or down.

The statistics for this domain can be seen in Table 5.4. This is an interesting domain, because even if it seems to be small and it only has one variable (the location of the agent), it has a high branching factor. Notice that the initial belief state corresponds with the agent not knowing anything about its locations.

Instance	$\|S\|$	$\|B\|$	$\|b_0\|$	Dead-ends	Actions
localize-5	25	625	625	0	8
localize-7	49	2401	2401	0	8
localize-9	81	6561	6561	0	8
localize-11	121	14641	14641	0	8
localize-13	169	28561	28561	0	8

Table 5.4: *Statistics for the 5 instances of Localize domain; localize−n corresponds to a grid $n \times n$.*

Instance	$\|S\|$	$\|B\|$	$\|b_0\|$	Actions
clog-7	387420489	1,500e+17	12	495
clog-huge	244140625000000	5,960e+28	3125	9475

Table 5.5: *Statistics for the 2 instances of Contingent Logistics domain. Total States indicates the total number of possible states in the problem, Total Belief States the total number of belief states, Size Initial Belief State is the number of possible initial states and Actions the total number of actions.*

5.1.5 Contingent Logistics Domain

Contingent Logistics (*Clog*) is the last problem from the set of benchmarks of the planner CLG. In this domain, packages are in unknown locations, and the agent must deliver them to their destination. *Clog* is a version of the logistic problem where all conditions of the effects have been moved as preconditions, and the original location of some packages is unknown. This domain is huge and difficult to solve because of the branching factor and the number of states. On the other hand, this domain has no *dead-ends*, and no As Figure 5.5 shows, the number of states is really huge. However the number of reachable states is smaller, because of the constraints on some locations and vehicles. And since most part of the state consist in fluents whose value is known, the number of Belief States is also reduced.

5.1.6 Doors Domain

This problem is a modification of the *Secret Doors* domain showed above. In this domain, instead of reaching the door by sensing the walls, the robot knows where the doors are. However the keys that open these doors are missing and the robot must find them before opening a door. The goal of the agent is also to reach the final room.

This domain is specified for contingent and probabilistic problems. The main difference lies in the fact that in the contingent version the agent does not know the initial location of the keys and must observe each location, and in a probabilistic setting where the agent must check every possible location, and there is a chance that the key is located at that position. Figure 5.2 shows an instance of this domain, with two different doors.

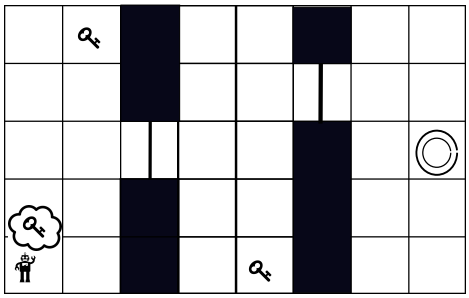


Figure 5.2: *Example of an instance of the Doors problem. The agent starts at a room, and must reach the goal room. To traverse to the other rooms must find the keys that open the doors. The initial location of the keys is not known.*

There exists *dead-ends* in this domain. In fact, there is a chance that the key may be missing of the room, for example trapped in another location inaccessible for the robot.

Figure 5.6 shows the statistics for this domain. Notice the difference between the Secret Doors

Instance	$\|S\|$	$\|B\|$	$\|b_0\|$	Dead-ends	Actions
doors-5-2	1800	2^{1800}	36	11	635
doors-5-4	233280	2^{233280}	1296	671	2045
doors-7-3	75264	2^{75264}	512	169	2422
doors-7-4	1032192	$2^{1032192}$	4096	1695	3997
doors-9-4	3240000	$2^{3240000}$	10000	3439	6597

Table 5.6: *Statistics for the 5 instances of Doors; doors−$n-m$ corresponds to a grid $n \times n$ with m doors.*

and the Doors domain. The *dead-ends* are very different, because in this domain a *dead-end* consists in a key located outside the reach of the robot.

5.2 Compact Compilation Belief Tracking Analysis

In this experiment, we tested our planner *Compact Compilation Belief Tracking* (COMP2BT, comparing it with the CLG planner (Albore et al., 2009) and to the PO-PRP planner (Muise et al., 2014), both considered to be the state-of-the-art contingent planning solvers, to evaluate the performance of the translation. We measure the efficiency of the planner as the time it takes to compute a complete plan, and the quality as the total number of actions. All experiments were performed in a Linux machine with an 1.33 Ghz processor. The times were limited to one hour for each one of the instances of every problem and to 1GB RAM memory.

To demonstrate where our translation overcomes the disadvantages of other translations, we also experiment with a variation with dead-ends for the Doors and Wumpus domain. In the Wumpus-D we removed the safe preconditions, and added an special effect that makes the agent die if it enters a cell that is not safe (i.e. it contains a Wumpus or a pit) creating a dead-end. This domain comes from the benchmark domains of the MPSR Planner Brafman and Shani (2012). In the new Doors-D domain, we added an effect that makes the robot break if it tries to enter through a door that is not opened, while removing the *opened* precondition.

Table 5.7 shows the comparison of the three planners CLG, PO-PRP and COMP2BT: For simple contingent planning problems (instances from wumpus-5 to doors-11), in general PO-PRP obtains the best results followed closely by COMP2BT, even if in some instances PO-PRP runs out of memory. For those problems, PO-PRP and COMP2BT are orders of magnitude better than CLG. Due to the fact that PO-PRP builds partial policies instead of trees, its plans have a better quality. A partial policy is a policy defined only for some *relevant* literals of the belief state, instead of a policy defined for all literals. In domains with dead-ends, COMP2BT is better than PO-PRP, being capable of solve more problems in a better time. Localize is a special domain, because of its uncertainty, and planners like PO-PRP have difficulties to deal with.

Thus, in domains with dead-ends, COMP2BT outperforms both PO-PRP and CLG, being capable of solve more problems in less time.

Linear translation One reason for COMP2BT to outperform CLG is that the later uses a quadratic translation based on tags and merges. Roughly speaking, this means that it considers every possible initial state, and includes in the translation new literals KL/t meaning that the literal L is known to be true if in the initial state t was true. Since in the CLG planner there is a tag for every possible initial state, the translated problem increases quadratically in size, and actions have more effects. As an example, a problem with 10 fluents translated with a linear translation can increase to $2 \times 10 = 20$ fluents. But with a quadratic translation it can grow to $(2 \times 10)^2 = 400$ fluents. The CLG planner takes more time updating belief states because it has to consider a larger number of epistemic fluents. COMP2BT and PO-PRP use a linear translation that results in a smaller translated problem with less literals. Both planners are better than CLG in all benchmarks. PO-PRP and COMP2BT use linear translation and get better results in most domains.

	CLG		COMP2BT		PO-PRP	
Problem	**Time**	**Size**	**Time**	**Size**	**Time**	**Size**
wumpus-5-3	1.28	754	0.23	435	0.48	197
wumpus-7-5	30.52	6552	2.97	4115	3.50	631
wumpus-10-8	T		76.81	63297	24.36	1471
wumpus-15-13	T		T		162.60	7787
cballs4-1	0.62	295	0.10	270	0.08	271
cballs4-2	60.84	20050	5.71	15149	5.54	12360
cballs4-3	T		448.89	892590	MO	
cballs10-1	337	4445	12.33	4799	5.02	3849
cballs10-2	T		T		MO	
clog-7	0.28	210	0.24	190	0.24	93
clog-huge	490.46	37718	39.99	28972	7.76	15799
secret-doors-7-3	18.76	2153	0.75	2241	0.68	1282
secret-doors-9-4	1294.64	46024	19.13	46656	18.30	23897
secret-doors-11-5	T		659.917	1208947	MO	
wumpus-d5	T		0.26	491	1.48	253
wumpus-d7	T		3.45	4589	17.86	947
wumpus-d10	T		88.56	69708	336.66	2497
secret-doors-d5-2	T		0.08	173	0.14	107
secret-doors-d7-3	T		1.14	2584	1.06	1304
secret-doors-d9-4	T		31.40	53217	28.62	22570
secret-doors-d11-5	T		MO		MO	
localize-5	0.4	115	0.09	106	2139.76	8399
localize-7	2.58	241	0.27	239	T	
localize-9	11.80	409	0.74	416	T	
localize-11	180.68	617	2.25	688	T	
localize-13	MO	MO	5.00	969	T	

Table 5.7: *Times in seconds and size of the policies obtained for the three different planners (*CLG, COMP2BT *and* PO-PRP*) to compute the full contingent plans. T stands for time-out and MO for memory out.*

Dealing with dead-ends In domains where actions may lead to dead-ends, like *wumpus-d* and *doors-d*, COMP2BT can detect these actions (see Section 4.4.6) and translate them in a version with no dead-ends, which effectively reduces the branching factor. However CLG fails to deal with this problems. CLG may fail when the local search (hill climbing) fails and then it has to restart with a complete heuristic search. Surprisingly PO-PRP can solve them, but with worse times than COMP2BT due to the discard of the policy when encountering a dead-end several times.

5.3 Human Help in Contingent Planning Problems

The HH-CP planner (*Human Help Contingent Planning*) (Andrés et al., 2017; Franch, 2017), is a planner that includes human help in the translation, in the form of literal actions compiled from the Causal Graph, as explained in Section 4.4.3. In this section we test the proposed algorithm over some contingent domains where no strong solution exists. We shows how our planner is able to resort to human help strictly when necessary.

Our HH-CP planner was tested in several domains, but in this section we focus in two: Wumpus World and Open-Doors, typically used to test contingent planners Albore et al. (2009) (See Section 5.3.4 for more tests). All experiments were performed in a Linux machine with an 2.4 Ghz processor. The times were limited to one hour and to 2GB RAM of memory, for each one of the instances of every domain. The efficiency of the planner is given in terms of time and the number of actions in the solution plan.

The goal of this empirical analysis is to show how our HH-CP planner deals with domains with different types of dead-end belief states, DEB1, DEB2 and DEB3, and their possible combinations.

5.3.1 Presence of DEB2 and DEB3 with Human Help Observations

In this experiment we only allow the planner to use human help observations. So, we expected the HH-CP planner to find strong plans only for instances with dead-ends of type DEB3. We also expect the plans will have few human help observations, given that we set cost $c_h = 50$ for the human

Instance	Time (ms)	Plan Size	HObs	Solution
5x5.1	0.40	61	5	Strong
5x5.2	0.08	16	1	Strong
5x5.3	0.08	16	2	Strong
5x5.4	1.50	228	5	Weak
5x5.5	0.22	37	1	Weak
7x7.1	0.20	20	1	Strong
7x7.2	0.22	30	1	Strong
7x7.3	0.48	42	1	Strong
7x7.4	0.22	26	1	Strong
7x7.5	4.78	465	6	Weak
10x10.1	0.38	27	1	Strong
10x10.2	1.30	109	4	Strong
10x10.3	1.80	126	3	Strong
10x10.4	3.76	175	1	Strong
10x10.5	7.18	352	4	Weak

Table 5.8: *Wumpus World problems with dead-ends of type DEB2 and DEB3, allowing human observations.*

Instance	Relevant Literals	Time (ms)	Size	Agent actions	Literal actions	Solution
5x5.4	138	160.88	238	9	1	Strong
5x5.5	138	272.00	44	6	1	Strong
7x7.5	226	27.50	472	6	1	Strong
10x10.5	434	427.25	362	9	1	Strong

Table 5.9: *Instances of the Wumpus World with dead-ends of type DEB2 allowing human help actions.*

observations while the agent's actions and observations have cost 1. We analyze the performance of the HH-CP planner in 15 instances of the Wumpus World domain (with randomly generated hazards), in terms of: time, plan size, number of human observations and type of solution (strong or weak). The plan size is the total number of all the actions, including human help observations.

Table 5.8 shows the results for: (i) 5 instances 5x5 (with 3 hazards, either Wumpus or pits, in 6 possible positions), (ii) 5 instances 7x7 (with 5 hazards, either Wumpus or pits, in 10 possible positions) and (iii) 5 instances 10x10 (with 8 hazards, either Wumpus or pits, in 16 possible positions). We noticed that 11 from the 15 instances have dead-ends of type DB3, since our planner generated strong contingent plans for them only with human observations. The 4 instances for which our HH-CP planner could not find a strong plan with human observations, contain dead-ends of type DEB2. Notice that for 3 of them, the costs and times are much higher. This is because the HH-CP planner always tries first to find strong policies and, only if no strong policy exists, it returns a weak policy. This process is time consuming and in general, returns larger plans. In sum, the results for this domain show that, HH-CP planner can efficiently solve problems with dead-ends of type DB3 using **only human observations**, while for problems with dead-ends of type DB2, it can be time consuming to generate strong solutions, as we show next.

5.3.2 Presence of DEB1 and DEB2 with Human Help Actions

In this experiment we solve problems with dead-ends of type DEB1 and DEB2 allowing only human help actions, and no observations. We selected the instances from the previous experiment for which the HH-CP planner generated a weak solution (Table 5.8). We used cost $c_l = 100$ for all human help actions. Table 5.9 shows that for all instances, the HH-CP planner needs to ask for human help only once to come up with a strong plan. However, since those are the larger instances, the number of human actions from the relevant set of literals is also large. Thus, HH-CP planner can spend a large time to find strong plans for problems with dead-ends of type DEB1 and DEB2 due to: (i) the greedy strategy of HH-CP planner and (ii) the large number of relevant literals used to create human actions.

Inst	HHCost	Time (ms)	Plan Size	HH
5x5	50	0.06	133	0
5x5	20	0.06	87	1
5x5	10	0.02	28	1
5x5	5	0.02	10	2
7x7	1000	5.76	2717	0
7x7	100	4.30	2302	1
7x7	50	2.90	1777	1
7x7	20	0.38	141	2
7x7	10	0.08	17	3
5x9	50	1,60	790	0
5x9	30	0.70	539	1
5x9	20	0.22	80	1
5x9	10	0.10	35	2
5x9	5	0.08	14	3
7x9	100	54.40	50402	0
7x9	50	4.58	2804	1
7x9	30	0.78	545	2
7x9	20	0.34	153	2
7x9	10	0.16	40	3
7x9	5	0.10	13	4
9x9	100	MO	MO	MO
9x9	50	3.96	1439	2
9x9	30	1.12	314	3
9x9	10	0.18	22	4

Table 5.10: HH-CP *performance in the Doors domain 5 instances with varying costs of human help (HH-Cost).*

5.3.3 Human Help to Minimize Plan Cost

In this experiment we show that even in a domain without dead-ends, an agent can decide to ask for human help in order to minimize cost. We used the *Open-Doors* domain, where a robot located in a grid must look for the keys to open the doors blocking its way to a goal location. There is only one key for each door. The robot can make a move to an adjacent cell and also sense a key located in its current cell, and open the door only after grabbing the corresponding key. The agent can also ask for human help (with a cost) to open a door, which will be necessary depending on the size of the problem, since the cost of looking for a key can be too high and the robot can decide that it is worth to call for help, especially when the contingent plan becomes too large. Thus, the only human help we allow in this domain is to open one or more doors.

Figure 5.10 (left) shows an example of a 5x8 instance with 2 doors of this domain. Figure 5.10 (Right) shows the table with the results for 5 instances of the Open-Doors domain, with varying costs of human help (HHCost), which are: 5x5 (with 2 doors), 7x7 (with 3 doors), 5x9 (with 4 doors), 7x9 (with 4 doors) and 9x9 (with 4 doors), whose size of the initial belief state is 25, 343, 625, 2401 and 6561, respectively. Note that all these instances have solutions that guarantee to reach the goal without human help, with the exception of instance 9X9 due to memory limitation (2GB). Notice that for this instance, with high human help cost (HHCost=100), the planning agent fails to generate a contingent plan due to memory out; however, with HHCost=50, it can solve this problem asking twice for human help. For all instances, as the cost of human help (HHCost) decreases, the agent chooses more often to ask human help, reducing the size of the plan and the planning time. Notice that the planning agent can generate a smaller plan asking for the same number of human help actions, e.g., for instance 5x9, when considering a human help of cost 30, the plan size is 539 and for cost 20, the plan size is 80, this happens because when the cost of the human action is lower, the agent applies this action before reducing the search tree. If the cost is higher, the agent tends to apply it later in the plan only when necessary, when other branches have been expanded.

	COMP2BT+HH				HH-CP			
Instance	**HH-actions**	**Time**	$\|\pi\|$	$\pi\_\#HH$	**HH-actions**	**Time**	$\|\pi\|$	$\pi\_\#HH$
clog-7	9	0.06	37	4	63	0.16	614	6
clog-H	25	0.40	43	5	-	MO	MO	MO
cballs4-1	4	0.08	259	1	54	0.752	348	16
cballs4-2	4	5.06	19144	51	92	6.270	15808	776
cballs4-3	-	MO	MO	MO	-	MO	MO	MO
cballs9-1	2	2.08	1493	3	249	3.14	3148	81
cballs9-2	-	MO	MO	MO	-	MO	MO	MO
cballs10-1	2	4.56	1953	1	306	20	4253	100
cballs10-2	-	MO	MO	MO	-	MO	MO	MO
doors5x5-2	2	0.1	167	12	119	0.12	297	11
doors5x9-4	4	1.7	4725	264	307	78.08	112480	2023
doors7x7-3	3	0.72	2059	80	279	6.38	12263	266
doors7x9-4	4	6.78	16290	592	-	MO	MO	MO
doors9x9-4	4	25.06	42526	1120	-	MO	MO	MO
wumpus-d5-3	-	MO	MO	MO	138	430.1	109	10
wumpus-d7-5	-	MO	MO	MO	226	82.28	484	18
wumpus-d10-8	-	MO	MO	MO	-	MO	MO	MO

Table 5.11: *Performance analysis of* COMP2BT+HH *and* HH-CP *planners in contingent domain instances with dead-ends. The cost of the human actions is the same for all problems. MO stands for memory out. Non unsolvable problems happen due to Time-out.*

5.3.4 Size of the set of Human help Actions

In the last experiment, we analyze how the size of the set of the human help impacts the performance of the planners, by comparing the COMP2BT+HH planner and the HH-CP planner. The main differences between these two planners is the set of human actions they use, and how they obtain them (see Sections 4.4.3 and 4.4.7 for details). Results show that a smaller set of human actions can result in better policies as well as in a better performance of the planner, however, there is no guarantee that this smaller set of actions can produce a strong policy and the planner may fail to find a policy if the human actions do not change any dead-end literal.

We tested both planners on modified versions of the original contingent problems. This modified versions contain unavoidable dead-ends, as explained for the versions of the *doors* and *wumpus* For the color balls (*cballs*) we included a ball with a different color, that does not correspond to any garbage. We created the possibility of the agent being unable to drop the ball. In the logistic problem (*clog*) the agent may find that the package is searching, is located in an isolated city without connections to the other cities, hence the agent is unable to create a plan to deliver this package.

Table 5.11 shows the human actions created (HH-actions), time to compute the policy (Time), the size of the computed policy ($|\pi|$) and the number of human actions used in the policy, considering all branches ($\pi\_\#HH$). Results show that, in general, when COMP2BT+HH is able to create the most appropriate actions, its plans are smaller and use less human actions (see Table 5.11). In the *cballs* problem, COMP2BT+HH correctly identifies that there exist *problematic* fluents, e.g. finding that a ball has a color that does not correspond to any garbage, and creates the appropriate actions (like throw the ball to an additional garbage). Likewise, in the *clog* domain, the agent is able to ask the human to take the package to a new location, creating a new connection between the isolated location and a location from where it is possible to find a strong solution. In the doors domain (*doors*), it creates the action that asks the human for the key to open a door. But in the Wumpus domain, where the dead-end may correspond to an state where the treasure and the Wumpus (or pit) are in the same location, it is only able to create the action that asks the human to make the agent alive again. Thus, the planner fails to detect that the dead-end is caused by the treasure being in the same cell as the hazard, and the agent ends stuck in a cycle even with human help. And since HH-CP works in two phases by computing a weak plan first and correcting it using the relevant fluents, it produces longer plans using more human help, generally using more time but it is safer since there exists always a relevant literal that can be changed to achieve the goal (Theorem 4).

5.4 Discussion

In this Chapter we have shown empirically how our approaches to solve contingent problems with dead-ends are not only sound, but also efficient when compared to the state of the art contingent planners. Belief tracking for planning with partial observation is an open and challenging problem in automated planning. We have also shown in this chapter how some causality relations among variables in a contingent planning problem can be exploited by a planner. We tested the proposed planner COMP2BT on 7 different contingent planning domains. The results show that our planner solved more problems than the two planners considered the state-of-the-art for contingent planning, presenting also better times (up to 2 orders of magnitude faster than CLG).

We have also shown how HH-CP is able to deal with contingent planning problems with dead-end belief states, where the agent is able to, proactively and autonomously, ask for human help: (1) to reduce the agent uncertainty in problems that have none strong solution; (2) to change the environment when there are pure dead-end states; and (3) find a cheaper solution when the cost without human help is higher than a given cost bound.

Results show that HH-CP planner can efficiently solve problems with dead-ends of type DB3, while to generate strong solutions for problems with dead-ends of type DB2, it can be time consuming, especially when the number of relevant literal-actions is large. In the presence of dead-ends of type DEB2, if the HH-CP planner takes an original weak solution that can only use human literal-actions to become strong, no matter how low a human observation costs, the HH-CP planner will always stick with a solution that includes human literal-actions.

Finally we have shown empirically, by comparing our planners COMP2BT+HH and HH-CP, how a smaller set of human actions may result in better policies using a minimal number of human help actions, as well as an improved performance of the planner.

Chapter 6

Related Work on Contingent Planning with Human Help

In this section we will focus on the symbiotic autonomy approaches to solve POMDP problems, and other approaches used in contingent planning to plan considering dead-ends.

Most of the work on human-robot collaboration in general (Karami et al., 2009; Schmidt-Rohr et al., 2008), and symbiotic autonomy in particular (Armstrong-Crews and Veloso, 2007; Rosenthal et al., 2011; Veloso et al., 2015) is based on POMDPs (*Partially Observable Markov Decision Process*), augmented with a set of human observations and actions, with negative reward and whose objective is to find a policy that maximizes the expected reward over a given horizon.

On the other hand, adjustable autonomy solutions usually model the problem as a MDP, where a high level controller divides the tasks between the human and the robot. An example of such approach is the *Mixed-Initiative Markov Decision Processes* (MI-MDP) (Côté et al., 2012; Mouaddib et al., 2009, 2010).

In all previous approaches, the human actions are known a priori, but the novelty in our work is the generation of these actions from the GMDP problem description. In this chapter, we will look closely to different approaches to model the human intervention and the reasoning about human help, and how these solutions compare to our approach in terms of: what kind of help the human can offer, when this help is asked and who is responsible for the completion of the task.

6.1 K-translations to deal with dead-ends

Incomplete translations do not work well in the presence of dead-ends. That is, a planner is unable to provide a strong policy, because they are incomplete (see Section 3.2.4).

However, there exist translations that using tags and merges are able to obtain weak policies more robust and that allow the agent more autonomy.

In contingent planning, CLG+ Albore and Geffner (2009) is an extenstion for the CLG planner, that deals with belief states B containing dead-ends by pruning these states from B, and then generate a plan from there. However, the apparently strong plan can fail if one of these dead-ends turns out to be the real state. In our approach, however, the planner does not eliminate any state from the belief state and never makes assumptions about the initial state. Rather than that, it plans and uses the human help whenever it cannot achieve the goal.

Another translation based approach to deal with dead-ends is the MPRS planner (*Multi Path Sampling Replanner*) (Brafman and Shani, 2012), that as we have seen in Section 3.2.4 are able to compute safer plans.

6.2 Symbiotic Autonomy

Due to the limitations of robot perceptions, it makes sense to rely on the superior sensitive capacities of humans. A robot can ask the human to perform observations, either perfect observations OPOMDP (*Oracular Partially Observable Markov Decision Process*) (Armstrong-Crews and Veloso, 2007) or or inaccurate observations HOP-POMDPs (*Human Observation Providers POMDPs*) Rosenthal et al. (2011). However this relation may be one-sided, if only the robots ask for help. In a realistic environment, situations where the robot and the human are helping each other to complete their respective tasks, can be more common. This kind of autonomy is called *Symbiotic Autonomy*, because each side helps the other to complete their different tasks.

In this section we present an overview of different approaches to solve a problem using human help actions as observations, or to overcome physical limitations of the robot and answering questions, providing observations and using physical force to move objects (*CoBot*).

6.2.1 Collaborative Robots

For example, the Collaborative Robot (*CoBot*) (Veloso et al., 2015) is an example of a robot designed to help visitors to reach their meeting locations. The human benefits from the robot knowledge of the different rooms and locations, while the robot also benefits from the human skills to overcome its limitations, such as physical help (open doors, lift cups, etc...). In the CoBot setting, the planning task is modeled as an MDP, where each state maintains information of the robot, such as its possible location in the environment. There are two categories of actions: *synchronous* (*Respond, Notify*) and *asynchronous* (*execute, inform, ask* and *request*). The synchronous actions require some degree of communication with the human, while the asynchronous could be performed if the preconditions hold. Both human and robot can perform asynchronous and synchronous actions. For example, a robot may inform the human of some locations of interest while moving. Or if the uncertainty about the robots location is increasing it may ask a human to help it locate itself, and when the human responds, the robot updates its location accordingly.

CoBots needs an initial policy π to start acting. This initial policy considers even actions that the robot is unable to do because of physical limitations. To model the robot limitations, these actions have *success* and *failure* effects. For example, *open-door* will always result in failure due to the lack of hands in the robot, and it must resort to human help. Modeling actions this way, it allows the robot to complete more tasks by asking for help. Moreover, aside from helping actions, CoBot can also ask humans to help locate itself, since CoBot does not know in which physical state is. And if the policy π to solve a task involves an action that the robot is not physically able to do, e.g., lift a cup of coffee, it must ask for help from the humans in the environment.

6.2.2 Oracular Observations

An Oracular Partially Observable Markov Decision Process (OPOMDP), is then a special kind of POMDP where the agent can not perform observations, relying in a *oracle*, human or sensor, instead. This oracle is capable of giving the exact information on a state, for a fixed cost. The oracle is always available, and does not depend on the position or the state of the agent to answer.

Formally, an OPOMDP is modeled like a POMDP, but any observation from the set of observations returns a null, indicating that the agent is unable to perform a single observation. However, there is an additional action o_h that provides the agent with full state knowledge. This action o_h represents a request to the human (or to a highly accurate sensor) for help, and in this setting is the only help the agent can obtain. This action also incurs in a higher cost $cost(o) = c_H$, so any agent that tries to minimize the expected cost will resort to this observation only when needed. Since the OPOMDP setting is defined for belief states, any action except the observation o, is deterministic. Because of the partial observation, a belief state is a set of states that are possible, hence, performing an action

with multiple (probabilistic) outcomes, will result in a belief state where each state represents one of these possible outcomes.

Solving an OPOMDP, involves computing an initial policy π, obtained without human help. This initial policy is obtained from the underlying MDP, that is without considering using human actions, and there is no need to modify the Bellman Equation, hence OPOMDP compute this solution using a Value Iteration based algorithm. However, during the execution and due to the uncertainty in the effects of the actions, the robots are allowed to ask for human help. A OPOMDP can be solved by the JIV algorithm, that works in two phases: (1) solve the underlying MDP, obtaining a complete policy π; and (2) use the solution computed before at execution time. During the second phase, the agent uses the calculated expected costs of every state to select which would be the best action to apply. To do so, it uses a weighted voting scheme, in which every state s of the belief state b select which is the best action to perform in that state, weighted by the probability of s being the real physical state $b(s)$. This value is compared to the expected cumulative cost of being in state s with *all certainty*, to decide if it is better to ask the oracle, obtaining this certainty, or to perform the action prescribed by the policy π. In other words, the agent executes the policy calculated π, except if the cost of asking is lower than the cost of executing under uncertainty.

Different from our approach, help in a OPOMDP is restricted only to observations. But in the presence of dead-ends, any agent may fail to reach the solution. Another difference is that solving a OPOMDP needs to be done in two phases: a planning phase that solves the underlying MDP, without human help, and the execution phase that uses a different algorithm that minimizes the expected cost during the execution.

6.2.3 Human Observation Provider

In a Human Observation Provider POMDP (HOP-POMDP), the human help is modeled in a more realistic way than in an OPOMDP. In an OPOMDP, the humans (oracle) are always available and the help is always accurate, however, this is not always the case. To reflect that, a HOP-POMDP models the help from humans in terms of their availability and accuracy Rosenthal et al. (2011). It is also assumed that humans are placed in known locations, and a human can help a robot only if they are in the same location.

Formally, a HOP-POMDP is modeled as a POMDP with three more elements: (1) an array to represent the cost of asking each human; (2) an array representing the availability of each human; and (3) an array representing the accuracy of each human. The cost of asking for help reflects the cost of interrupting a human, and the time it may take to answer. The availability of a human is defined as the probability that a human provides a non-null observation. And the accuracy is defined as the probability that the observation the human provides is the correct observation.

Also, in a similar way to a OPOMDP, solving a HOP-POMDP depends on having solved the MDP problem beforehand, and obtaining a policy π. And then, an online algorithm learns the availability and accuracy for each human. If the policy prescribes the agent to ask help in a state s from a human h, but the human is not available, then the agent must apply the best non-asking action, as dictated by the policy π.

The online learning algorithm to solve HOP-POMDP, called LM-HOP (Learning the Model of Humans as Observation Providers) has three main functions: (1) compute a single HOP-POMDP solution π obtained from the underlying MDP; (2) learn and update human accuracy and availability; and (3) selective learning.

In any state s before applying the action $\pi(s)$, the agent chooses randomly if it must execute action $\pi(s)$, or a random action a. After this step, the agent updates the belief state as usual, but if the random action is a human observation (an *ask* action), then it must update the availability and accuracy, averaging the observation received by the probability of being in that state $b(s)$. But in order to reduce the number of times a policy is recomputed, the algorithm only updates the policy when the estimated availability or accuracy changes significantly from the current estimate.

This model is a bit more realistic than OPOMDP, however it allows human help only for observation. The algorithm is devised mainly to obtain the true values of availability and accuracy of the

humans in the environment. Unlike our approach, it needs to compute a policy for the underlying POMDP model, to learn online the true availability of the agent.

Part II

Human Help in Probabilistic Planning

In the previous chapters we have described fully observable (FOND) and partially observable non-deterministic planning problems (POND) and how to obtain strong policies from problems with dead-ends, by using human help. In the next four chapters we shift our focus from POND problems to full-observable probabilistic planning (FOPP), that is, problems where a probability can be used to reflect how an effect of an action is likely to occur.

We start in Chapter 7 by formally defining probabilistic problems, starting with MDP (Section 7.1.1), one of the most studied classes of probabilistic problems. We will follow with a subclass of MDP problems, called goal-oriented markov decision processes (GMDP) (Section 7.1.1), and continue with a more restrictive subclass called *stochastic shortest path* (SSP) (Section 7.1.2). We also explain how dead-ends affects the solutions of these problems (Section 7.3), and conclude this chapter by describing current approaches to deal with dead-ends (Sections 7.3.2 and 7.3.3).

In Chapter 8 we will introduce a generalization of goal-oriented markov decision processes (GMDPs), called **GMDPs augmented with human help (GMDP-HH)** (Section 8.1), where a robot has a distinguished set of human help actions that can be applied in any state. We consider human help to be a costly resource to be used parsimoniously. This introduces a dilemma for the robot: to maximize the probability of reaching the goal (possibly maximizing the probability of asking for human help), or to minimize the probability of human interaction (possibly maximizing the probability of reaching a dead-end). We address this dilemma by recasting a GMDP-HH as a MAXPROB problem (Section 8.1.1) and show the shortcomings of this approach. Then, we propose different optimization criteria (Sections 8.2 and 8.3) and algorithms to solve GMDP-HH.

Next, in Chapter 9, we evaluate the scalability of our proposal and investigate the characteristics of policies obtained with different criteria in augmented versions of three standard planning domains (Doors, Tire World and Navigation). The experiments suggest that minimizing expected cost with or without the penalty is an effective approach for reasonably large problems. Interestingly, increasing the cost and penalty of human actions improves the quality of the heuristics, which in turns increases convergence of the methods.

Finally, in Chapter 10 we review the related work, like *adjustable autonomy* approaches (Section 10.1), as well as other approaches to solve these problems (Section 10.2).

Chapter 7

Probabilistic Planning: background

In previous chapters we have discussed dead-ends in planning problems with partial observable environments and non-deterministic actions. We have also shown how to consider human help and translate these problems to FOND problems, to be solved by the PRP planner. In this chapter we formalize probabilistic planning problems as *goal oriented Markov decision processes*.

7.1 Markov Decision Processes

In a probabilistic planning problem, probabilities may be associated either to the action effects or the observability. In this work we focus on probabilistic planning problems that are modeled as a *Markov decission process* (MDP) (Puterman, 1994) with full observability and stochastic actions.

A *Markov decision process* (MDP) models the interaction between the agent and the environment. In each step t, the agent in state s_t selects, an action a_t that minimizes an utility function. After applying action a_t, the agent is in state s_{t+1}, and receives a cost (or reward) c_{t+1}. In general, the utility function depends on the accumulated cost of the applied actions. Thus, an MDP can be seen as an optimization problem, where the objective is to minimize the accumulated cost in the interaction between the agent and the environment during a given number of steps, called horizon. Three different types of horizon are considered:

- Infinite: in this case the agent acts for an infinite number of steps, that is, it never stops acting;
- Finite: where the agent acts for a predefined number of steps; and
- Indefinite: where the agent acts in the environment until reaching a goal state or a dead-end.

Since planning problems involve initial state and goal states, in this work we focus on goal-driven MDPs, called *goal oriented Markov decision processes* (GMDP), which have an indefinite horizon.

7.1.1 Goal Oriented Markov Decision Processes

A GMDP is an MDP augmented with an initial state s_0 (or a set of initial states S_0) and a set of goal states G:

Definition 23 (Goal-Oriented MDPs (GMDPs))**.** *A* GMDP *is a tuple* $\mathcal{M} = \langle S, s_0, S_G, A, \mathcal{P}, \mathcal{C}, \rangle$ *where:*

- S *is a fully observable set of states;*
- s_0 *is the initial state;*
- $S_G \subseteq S$ *is a set of absorbing goal states, that is:*

$$\forall (a, s) \in A \times G, \mathcal{P}(s, a, s) = 1 \text{ and } \mathcal{C}(s, a) = 0;$$

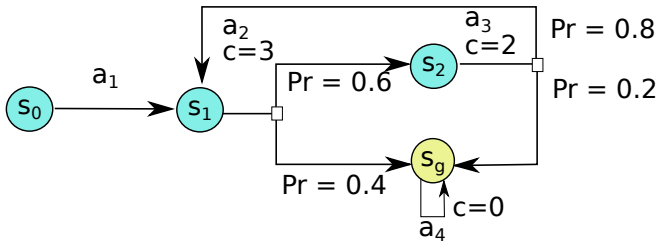


Figure 7.1: *Example of a* GMDP*, represented as a transition state graph. Nodes are labeled as states s_i; edges are labeled as actions a_i; deterministic transitions are directed edges with probability 1; probabilistic transitions are represented by edges branching from a box $\square$, each branch with $Pr < 1$. For clarity, unary costs are omitted.*

- *A is a set of actions, and we define $A(s)$ as the set of actions applicable in the state s;*
- *$\mathcal{P} : S \times a \times S \to \mathbb{R}$ is the probabilistic state transition function, such that $\mathcal{P}(s, a, s')$ indicates the probability that applying action $a \in A$ in state $s \in S$ will result in the agent being in state $s' \in A$; $\mathcal{P}(s, a, s') = 0$ if $a \notin A(s)$;*

$\diamond$

The solution of an GMDP is a policy defining the behavior of the agent, that minimizes the expected accumulated cost. Notice that all actions applied in any goal state has a cost of 0, and the agent remains in the goal state. As we will see, modeling goal states as absorbing states is suitable for using dynamic programming algorithms such as *Value Iteration* (VI).

Definition 24 (Policy)**.** *A **policy** $\pi : S \to A$ is a mapping from states to actions that prescribes the agent behavior. A policy can be complete if is defined for all states $\forall s \in S$ or partial, that is $\forall s \in S' \subset S$* [1] $\diamond$

Figure 7.1 shows a directed graph where labeled nodes represent states and edges represent actions. State s_1 represents the initial state and state s_g represents the goal. Notice that the only action applicable in the goal state s_g takes the agent to the same state s_g with cost 0.

Language Representation . A GMDP can be more concisely and conveniently specified using an extended set-theoretic STRIPS extended to accommodate probabilistic effects, where states are represented in terms of **fluents**, predicates (properties) whose truth value can be modified by the actions of the agent.

Definition 25 (Set-theoretical GMDP problem)**.** *A* GMDP *expressed in a language of actions is a tuple $\mathcal{M}_F = \langle F, I, G, \mathcal{A} \rangle$, where F, I and G are as in Definition 2, and:*

- *$\mathcal{A}$ is a finite set of triples $\langle prec(a), eff(a), cost(a) \rangle$ of preconditions, effects and cost respectively.*

$\diamond$

By making a closed world assumption, any state $s \in S$ can be identified with the set of propositions that hold true in that state. The actions, probabilistic transition functions and costs are jointly represented as probabilistic **planning operators** of the form $a = \langle prec(a), eff(a), cost(a) \rangle$, where $prec(a) \subseteq F$ is a set of **preconditions** that must hold true so that a can be applied, $cost(a) \in \mathbb{R}^+$ is the cost of executing the action, and $eff(a)$ is a finite set of probabilistic effects. Probabilistic effects are also a tuple $\langle p, add(e), del(e) \rangle$, where $add(e) \subseteq F \setminus prec(a)$ and $del(e) \subseteq prec(a)$ are sets of propositions denoting, respectively, which propositions are set to true/false after applying the effect, and $p \in [0, 1]$ is the probability that the effect occurs (the sum of all such p's must add to

[1] We implicitly assume that every state has at least one applicable action.

one for any action). An action a is **applicable** in state s if $pre(a) \subseteq s$. An action a applicable in state s takes the agent to a state $s' = (s \setminus del(e)) \cup add(e)$ with probability p and cost $cost(a)$. And the set of successor states of s, S' is the set:

$$\bigcup_{\forall e \in E(a)} (s \setminus del(e) \cup add(e)).$$

We assume that if a is not applicable in s then $\mathcal{P}(s, a, s') = 0$ for any state s'.

A GMDP is a model to describe the world. It is still necessary to define an optimization criterion to obtain optimal policies. To do so, it is necessary to make some assumptions, for example: *it is possible to reach a goal state from all states $s \in S$*; *it is possible to reach the goal from the initial state s_0*; *policies that do not reach the goal have an infinite cost.* This different assumptions create different classes of GMDP problems as we see next.

7.1.2 Stochastic Shortest Path MDP

Definition 26. *A Stochastic Shortest Path* MDP *(*SSP-MDP*) is a* GMDP $\mathcal{M} = \langle S, s_0, G, A, \mathcal{P}, \mathcal{C} \rangle$ *that makes two restrictive assumptions (Bertsekas and Tsitsiklis, 1991):*

(S1) *it is possible to reach a goal state from all states $s \in S$; and*

(S2) *policies that do not reach the goal have an infinite cost.*

◇

The objective of a SSP-MDP is to find a policy that minimizes the expected accumulated cost of the actions that lead the agent to the goal states. To formalize the assumptions of a SSP-MDP we should explain some concepts first.

A *history* $\sigma = \langle s_1, s_2, \ldots, s_k \rangle$ is a sequence of states visited in k steps, when following a policy π. The probability that the story σ occurs when executing policy π is given by:

$$P_\sigma^\pi = \prod_{i=0}^{|\sigma|} \mathcal{P}(s_i, \pi(s_i), s_{i+1}), \tag{7.1}$$

where s_i and s_{i+1} denote the states visited in steps i and $i+1$ respectively; and $\pi(s_i)$ denotes the action taken in state s_i according to policy π.

The expected accumulated cost of a history $\sigma = \langle s_1, s_2, \ldots, s_k \rangle$ following policy π is given by the expression:

$$V^\pi(\sigma) = \sum_{i=0}^{k-1} C(s_i, \pi(s_i)); \tag{7.2}$$

Now, let $S' \subseteq S$ be an arbitrary set of states, and let $\mathcal{H}_s^{S'}$ the set of histories that start in a state s and end up in a state $s' \in S'$. We define the probability of reaching a state $s' \in S'$, from a state $s \in S$ following a policy π, denoted by $P_\pi^{S'}(s)$, as the sum of the probabilities of each history of $\mathcal{H}_s^{S'}$, that is:

$$P_{S'}^\pi(s) = \sum_{\sigma \in \mathcal{H}_s^{S'}} P_\sigma^\pi. \tag{7.3}$$

where P_σ^π is the probability of history σ occurring while following policy π (Eq. 7.1).

We also define the expected cost of a policy reaching a state $s' \in S'$, from a state $s \in S$ as the sum of the expected accumulated cost of a history weighted by the probability of the history:

$$J^\pi = \sum_{\sigma \in \mathcal{H}_s^{S'}} V^\pi(\sigma) P_\sigma^\pi, \tag{7.4}$$

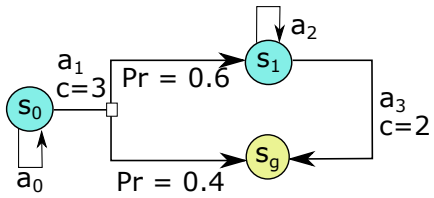


Figure 7.2: *Example of a* GMDP *with a proper policy ($\pi_1 = \{s_0 : a_1; s_1 : a_3\}$) and improper policies ($\pi_2 = \{s_0 : a_1; s_1 : a_2\}$ and $\pi_3 = \{s_0 : a_0; s_1 : a_3\}$). Unary costs and actions in goal states are omitted for simplicity.*

And the expected cost of any history that reaches an state $s' \in S'$ from state s following a policy π, denoted as $V^{\pi}_{S'}(s)$ is defined as:

$$V^{\pi}_{S'}(s) = \sum_{\sigma \in \mathcal{H}^{S'}_s} V^{\pi}(\sigma) P^{\pi}_{S'}(\sigma_s), \tag{7.5}$$

i.e. $V^{\pi}_{S'}(s)$ is the sum of the costs (Equation 7.2) of the histories $\sigma \in \mathcal{H}^{S'}_s$ weighted by their probabilities (Equation 7.1). A policy π is s-proper if it has a probability 1 of reaching the goal from the state $s \in S$, i.e., $P^{G}_{\pi}(s) = 1$.

Proposition 3. *Given a state $s \in S$, if a policy π is s-proper, then for all states s' reachable following π from s, denoted by the set $Reach(s, \pi)$, π is also s'-proper. That is, if $P^{\pi}_{G}(s) = 1$, then $\forall s' \in Reach(s, \pi)$: $P^{\pi}_{G}(s') = 1$*

Proper Policy A policy π is *proper* if for all states $s \in S$, π is s-proper. In other words, a policy π is proper if an agent following π reaches the goal from all states, that is $\forall s \in S : P^{\pi}_{G}(s) = 1$. On the other hand, a policy π is said to be *improper* if it is not a proper policy, i.e., there exists at least one state s with a probability of reaching the goal $\exists s \in S : P^{\pi}_{G}(s) < 1$.

In Figure 7.2 it is shown a GMDP with a proper policy $\pi_1 = \{s_0 : a_1; s_1 : a_3\}$, and improper policies π_2 and π_3. Notice that following policy π_1 the agent will always reach the goal, i.e $P^{\pi_1}_{G}(s_0) = 1$. On the other hand, policy π_2 has a probability 0.4 of reaching the goal $P^{\pi_2}_{G}(s_0) = 0.4$, and probability 0.6 of leading the agent to state s_1 and remain in it, therefore π_2 is an improper policy. Notice also that an improper policy as π_3 can be s_1-proper, because even if $P^{\pi_3}_{G}(s_0) = 0.4$ applying policy π_3 in state s_1 will result in the agent reaching the goal.

Proper policies always reach the goal eventually, hence their expected accumulated cost is finite. If we consider that cost-functions are always positive, then improper policies will have an infinite cost. However, this assumption is not enough as some cost functions can be zero or even negative, so the expected accumulated cost can be a finite number, or even infinite negative. Formally, the initial assumptions **S1** and **S2** can be written as:

(S1) $\exists \pi : P^{\pi}_{G}(s) = 1, \ \forall s \in S$, i.e., there exists at least one proper policy π; and

(S2) $\forall \pi :$ if $\exists s \in S$ s.t. $P^{\pi}_{G}(s) < 1$ then $J^{\pi} = \infty$, i.e., all improper policy has an expected cost of ∞, as given by Equation 7.2.

The GMDP of Figure 7.2 is an example of an SSP-MDP. Policy $\pi_1 = \{s_0 : a_1; s_1 : a_3\}$ is a proper policy, that satisfies assumption **S1**, and since all actions have positive cost, the expected accumulated cost of the improper policies is ∞.

Optimal Policy The optimal policy of a SSP-MDP $\mathcal{M}$ is the proper policy that minimizes the expected accumulated cost of the histories that reach the goal (Equation 7.5):

$$\pi^* = arg \min_{\pi \text{ is proper}} V_G^{\pi}(s), \forall s \in S. \tag{7.6}$$

The optimal policy π^* of a SSP-MDP is the greedy policy associated to the value function $V^*(s) = min_\pi V^\pi$, that satisfies the optimality equation of Bellman (Bertsekas and Tsitsiklis, 1991):

$$V^*(s) = \begin{cases} 0, & \text{if } s \in G; \\ \min\limits_{a \in A} \left\{ \mathcal{C}(s,a) + \sum\limits_{s' \in S} \mathcal{P}(s,a,s') V^*(s') \right\} & \text{otherwise.} \end{cases} \tag{7.7}$$

Using dynamic programming, it is possible to compute the solution of a SSP-MDP. The *Value Iteration* algorithm can solve a SSP-MDP using Equation 7.7 as an attribution function to the value function:

$$V_{t+1}(s) \leftarrow \min_{a \in A} \{ \mathcal{C}(s,a) + \sum_{s' \in S} \mathcal{P}(s,a,s') V_t(s') \}, \tag{7.8}$$

initializing with arbitrary values $V_0(s)\ \forall s \notin G$, and $V_0(s) = 0\ \forall s \in G$.

The *Value Iteration* algorithm (Bertsekas and Tsitsiklis, 1991) updates the value function of *all* states in each iteration, and for this reason is called a *Synchronous Dynamic Programming Algorithm*. When this algorithm converges, it has created a policy π defined for all states $s \in S$. However this result in excessive calculation, and for larger problems this algorithm is not efficient, because of the exponential growth depending on the number of the atomic propositions of the problem, i.e., $|S| = 2^{\mathbb{P}}$, where $\mathbb{P}$ is the number of atomic propositions of the problem. In the following section are described some *asynchronous* algorithms that compute policies only for the reachable states from the initial state s_0.

7.2 Asynchronous Probabilistic Planning Algorithms

We have explained how synchronous methods, such as *Value Iteration* solve SSP-MDP updating all states in each iteration. In this section we present first a different and more efficient strategy, called *Find-and-Revise*, that updates only one state at each step. Then we present an algorithm implementing this strategy, called *Labelled Real-Time Dynamic Programming*.

The strategy *Find-and-Revise* (F & R) (Bonet and Geffner, 2003) is a general way to solve SSP-MDP asynchronously. This strategy is based on the two assumptions of SSP-MDP **S1** and **S2** to guarantee that an optimal policy is found in a more efficient manner. To do so, F & R initializes the value function with an admissible heuristic, and updates only one state in each iteration, thus reducing the convergence time of the algorithm. And when the residual error of all states appearing in the greedy graph rooted in the initial state is lower than a value ϵ, this algorithm converges. Bonet (2003) showed that this strategy converges when the heuristic used is admissible.

Algorithm 2 *Find-and-Revise* strategy (Bonet and Geffner, 2003)

INPUT: SSP-MDP task $\mathcal{M}$
OUTPUT: Optimal value function V^* of $\mathcal{M}$

```
1: function FIND-AND-REVISE((M))
2:     Initialization V with an admissible heuristic
3:     repeat
4:         Find states s in the Greedy Graph G_V with Res(s) > ε
5:         Revise V from the states s
6:     until ∀s ∈ G_V Res(s) > ε
7:     return V
8: end function
```

Given an SSP-MDP $\mathcal{M}$, the strategy F & R use the greedy Graph G_V, rooted at s_0 to find the

states not still converged that need to be updated, thus computing the value function V only for reachable states from s_0 and creating a partial greedy policy π defined only for a set of states $S' \subseteq S$ such that $S' = Reach(s_0, \pi)$. As *Value Iteration* algorithm, F & R computes the optimal value function V^* with maximal residual error ϵ.

Algorithm 2 shows a description of the F & R strategy. In line 2 the value function V is initialized using an admissible heuristic. Then the F & R searches in the greedy graph for the states with a residual larger than ϵ (line 4), revising them, that is, updating their value (line 5) until there is no state in the Greedy Graph G_V with a residual larger than ϵ (line 6).

The efficiency of the algorithms that implement this strategy depends on the heuristic used, and on the order in which the states are updated. The closer the heuristic value of an state s is to the optimal value V^*, the fewer the number of updates this state s will require before convergence. In the next sections we present algorithms that implement this strategy.

LRTDP The *Real-Time Dynamic Programming* (RTDP) algorithm (Barto et al., 1995) is a probabilistic planning algorithm that selects the set of states to be updated from the initial state s_0 by *samplings*, called *trials*. A *trial* simulates the execution of the greedy policy π from the initial state in the following way: given an state s, the agent then selects a greedy action a and selects one state s' from the set of successor states, according to the probability $\mathcal{P}(s, a, s')$. This operation is repeated until a goal state is found. RTDP updates the value function of an state at each iteration, using the Bellman equation (Equation 7.8), as the *Value Iteration* algorithm.

The *Labeled* RTDP (LRTDP) (Bonet, 2003) algorithm is an extension of the RTDP, that follows the strategy F & R. This algorithm labels states already considered converged, and stops to update the value function of these states accelerating the convergence.

In algorithm 3 the procedure LRTDP receives an initial state s_0 and performs *trials* until s_0 is labeled as solved, which means that the value function converged for all states in the greedy graph G_V. The LRTDP-TRIAL procedure performs a trial in the following way: receives an state, and selects the greedy action to be executed (line 13), updates the value function (line 14) and then picks the next state stochastically simulating an interaction with the environment in the method PICK-NEXT-STATE (line 15). The trial continues visiting states until reaching either a goal state or a state labeled as solved. If a visited state s has not been initialized, LRTDP calls the procedure INITIALIZE, that uses heuristic h to initialize the value of s. Notice that each visited state is stacked in *visited*. At the end of each trial, the states s_i are treated in reverse order, and the method CHECK-SOLVED verifies if all states of the greedy graph rooted at s_i have already converged, and in that case it labels the state s_i as solved, but if s_i has not yet converged, CHECK-SOLVED discard the rest of states (line 15). The idea behind this procedure (CHECK-SOLVED) is that if s_i is not converged, its predecessors in the graph have also not converged. The length of trials diminishes as the states are labeled, so not solved states will be visited more frequent effectively reducing the time to converge of LRTDP, compared to RTDP. Since the simulation with the environment will select the states with more probability of being visited, these states will converge faster. This will produce *anytime* policies, meaning that at any time of the execution is possible to obtain a useful policy.

7.3 Dealing with Dead-Ends

GMDP can have dead-ends states from where it is not possible to reach the goal. For example, consider Figure 7.3 where state d_0 is a *dead-end* because there is no policy that can lead the agent to the goal from that state.

Since in a *dead-ends* s, by definition there is no possibility of reaching the goal, that is, there is no s-proper policy, a GMDP with at least one *dead-end* is not a SSP-MDP because it does not satisfy the first assumption **S1**.

Another problem is where a problem contains cycles of zero or negative cost, that violates the second assumption **S2**, because in a state an agent could stay in a cycle where the expected

Algorithm 3 LRTDP Algorithm (Bonet and Geffner, 2003)

INPUT: SSP-MDP task $\mathcal{M}$
OUTPUT: Optimal value function V^* of $\mathcal{M}$

```
1: function LRTDP((M))
2:     while ¬s_0.SOLVED do
3:         LRTDP-TRIAL(s_0)
4:     end while
5: end function
6: procedure LRTDP-TRIAL(s_0)
7:     visited=EMPTY-STACK                          ▷ Stacks s in visited
8:     while s ∉ G and ¬s.SOLVED do
9:         visited.PUSH(s)
10:        if V of s is not defined then
11:            INITIALIZE(s, h)
12:        end if
13:        a=GREEDY-ACTION(s)                       ▷ Chooses greedy Action
14:        s.UPDATE(s)                              ▷ Updates value function of state s
15:        s = PICK-NEXT-STATE(s, a)                ▷ Stochastically simulate next state
16:    end while
17:    while visited ≠ EMPTY-STACK do               ▷ try labeling visited states in reverse order
18:        s = visited.POP()
19:        if ¬ CHECK-SOLVED(s) then
20:            break
21:        end if
22:    end while
23: end procedure
24: procedure INITIALIZE((s, h))
25:    V(s) = h(s)
26: end procedure
```

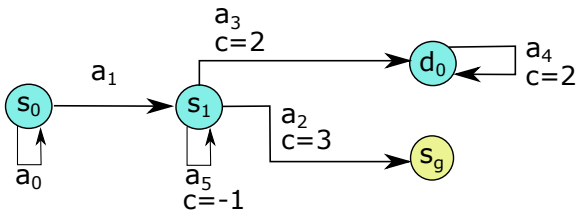


Figure 7.3: *Example of a* GMDP *with dead-ends (state d_0) and negative cost cycles (in state s_1). Unary costs and actions in goal states are omitted for simplicity.*

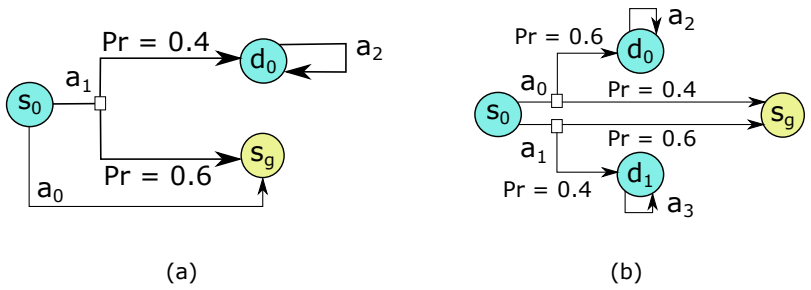


Figure 7.4: *Example of* GMDP *problems with avoidable and unavoidable dead-ends. (a)* GMDP *with avoidable dead-ends; and (b)* GMDP *with unavoidable dead-ends. Unary costs and actions in goal states are omitted for simplicity.*

accumulated cost is zero, or infinite negative. As an example consider the cycle in state s_1 induced by the action a_5 of Figure 7.3.

The algorithms showed in this chapter will only work if the assumptions **S1** and **S2** hold, when solving problems of indefinite horizon without discount factor. It is necessary, then new models allowing the agent to decide what to do in these case. For example the agent might want to maximize the probability of reaching the goal, or it could use a discount factor, guaranteeing that the expected accumulated cost is finite.

7.3.1 Goal Markov Decision Process with Dead-Ends

Definition 27 (Dead-end)**.** *Given an* GMDP $\mathcal{M} = \langle S, A, \mathcal{P}, \mathcal{C}, s_0, G\rangle$*, a state $s \in S \setminus G$ is a dead-end if, and only if, the probability to reach a goal state $s' \in S$ from s is* 0*, that is,* $\forall \pi : P^{\pi}_{G}(s) = 0$ $\diamond$

There exist many types of dead-ends. In Kolobov et al. (2010) dead-ends are divided in two main categories: *explicit dead-ends* and *implicit dead-ends*. Explicit dead-ends correspond to absorbing states, i.e. no action takes the agent out of those states. And implicit dead-ends correspond to states that, while they are not absorbing states, there is no path from an implicit *dead-end* to the goal state, and every state reached from an implicit *dead-end* is also a *dead-end*. Another important category of dead-ends are the *traps*, consisting in strong connected components in the planning graph with 0 probability of reaching the goal, and that once the agent enters, it is unable to escape.

A more interesting classification concerning planning problems with dead-ends, consist in dividing the problems in two classes: SSP-MDP with *avoidable* dead-ends and *unavoidable* dead-ends (Kolobov et al., 2012). A problem with *avoidable* dead-ends is a problem where there exists at least one s_0-proper policy, and a problem with *unavoidable* dead-ends is a problem where there not exists an s_0-proper policy. Figure 7.4 shows an example of each class of problem. In the SSP-MDP of Figure 7.4(a) there exists one proper policy for the initial state, that is $\pi_1 = \{s_0 : a_0\}$. But in the SSP-MDP of Figure 7.4(b) it does not exists a proper policy for the initial state, because both actions may end in a state where a goal state is not reachable. However, it must also be noted that *avoidable* or *unavoidable* dead-ends are a characteristic of the problem, not of the dead-end itself.

Kolobov et al. (2012) define these two new classes of MDP where the **S1** and **S2** assumptions for the SSP-MDP do not hold, allowing the existence of dead-ends, namely SSP-MDP with *avoidable* dead-ends and *unavoidable* dead-ends. They also define new optimization criteria to solve these problems that we will describe next.

7.3.2 SSP-MDP with *Avoidable* Dead-ends

As seen in previous sections, an SSP-MDP is restricted to those problems where there exists a proper policy for each state (assumption **S1**). But as explained above, this assumption is too restrictive and interesting problems fall out of this category. The first relaxation for these assumption is to consider that even if dead-end exist, there exists at least an s_0-proper policy:

Definition 28 (SSP-MDP with avoidable dead-ends)**.** *An* SSP-MDP *with avoidable dead-ends, called* SSPADE *(Kolobov et al., 2012), is an* GMDP $\mathcal{M} = \langle S, A, \mathcal{P}, \mathcal{C}, s_0, G \rangle$ *and:*

(A1) there exists at least a policy π*, such that* $P_G^\pi(s_0) = 1$ *, i.e., there exists at least a* s_0*-proper policy; and*

(A2) $\forall \pi$ *s.t.* $P_G^\pi(s_0) < 1$*, then* $J^\pi(s_0) = \infty$*, with* $J^\pi(s_0)$ *given by Equation 7.4.*

◇

The optimal policy π^* that solves an SSPADE is the s_0-proper policy that minimizes the expected accumulated cost of histories that start at s_0, and reach a goal state $s_g \in G$:

$$\pi^* = arg \min_{\pi \text{ is } s_0-proper} V_G^\pi(s) \tag{7.9}$$

As an example consider the problem showed in Figure 7.4(a), where there exist two different policies $\pi_1 = \{s_0 : a_0, d_0 : a_2\}$ and $\pi_2 = \{s_0 : a_1, d_0 : a_2\}$. π_1 is a s_0-proper policy, satisfying assumption **A1**, because $P_G^{\pi_1}(s_0) = 1$ (notice that π_1 is not a proper policy because $P_G^{\pi_1}(d_0) = 0$). But π_2 is not a proper policy $P_G^{\pi_2}(s_0) = 0.6 < 1$. Also notice that since there is a cycle in state d_0, induced by action a_2, the expected accumulated cost of π_2 $J^{\pi_2}(s_0) = \infty$, and assumption **A2** is also satisfied, hence the problem showed in Figure 7.4(a) is an SSPADE.

An interesting property derived from this assumptions is that SSP-MDPs are SSPADE.

Proposition 4. *An* SSP-MDP*s is an* SSPADE*.*

Proof. SSP-MDPs satisfy both assumptions **A1** and **A2** of SSPADE. □

Algorithms that solve SSP, like *Value Iteration*, may not work for a SSPADE, because of the relaxation on the assumption **S1**:

Value Iteration Even if there exists a s_0-proper policy, dead-ends still exist, and they have an expected accumulated cost of ∞. Since VI algorithms compute the expected accumulated cost for **all** states in the state space, this algorithm never converges, and the expected value $V(s)$ increases at each iteration. Even if we restricted the update of the expected cost $V(s)$ only for the reachable states, this would not help because SSPADE do not exclude dead-ends reachable from s_0. To use a Value Iteration approach it should be necessary first to identify the dead-end states.

Heuristic search Algorithms using the *Find-and-Revise* strategy (F & R) 7.2, are able to solve SSPADE. Since an heuristic search starts by assigning an estimate to all states, and at each step it updates the expected accumulated cost $V(s)$ of a state s, the costs of dead-ends $V(s_{de})$ will never stop increasing, while expected costs of other states will converge to finite values This will cause the greedy search to never select dead-ends, and any algorithm using a F & R strategy will end computing an s_0-proper policy. Notice that even if the algorithm LRTDP implements the strategy F & R, if a trial reaches a dead-end state, as in state d_0 in the problem showed in Figure 7.4(a), it can stay in that dead-end forever. A simple way to overcome this, is to never visit repeated states, ending the trial when a repeated state is found, or as the implementation of *lrtdp* does, setting a threshold to the maximum value a state can achieve and ending the search when the expected accumulated cost reaches this value.

7.3.3 SSP-MDP with *Unavoidable* Dead-ends

The second class of problems proposed by Kolobov et al. (2012), denoted by SSPUDE (SSP-MDP with *unavoidable dead-ends*), considers SSP-MDP problems to have *unavoidable* Dead-ends. If every policy that is not proper incurs in an ∞ expected cost, no planner will be able to distinguish between improper policies, rendering the criteria of minimizing the cost useless. As an example consider two

improper policies for the problem showed in Figure 7.4(b) $\pi_1 = \{s_0 : a_0, d_0 : a_2, d_1 : a_3\}$ and $\pi_2 = \{s_0 : a_1, d_0 : a_2, d_1 : a_3\}$. They have a probability $P_G^{\pi_1}(s_0) < 0.4$ and $P_G^{\pi_2}(s_0) < 0.6$, and their expected costs $J^{\pi_1}(s_0) = \infty$ and $J^{\pi_2}(s_0) = \infty$. According to the criterion of minimizing the expected accumulated cost from s_0, both policies are indistinguishable, that is, both have the same expected accumulated cost and the criterion is not helpful to decide which policy is better. The intuition says that policy π_2 is better, because it has a higher probability of reaching a goal state. Thus, we need new criteria able to distinguish between these situations.

Kolobov et al. (2012) proposes two new criteria based on paying a positive penalty $D \in \mathbb{R}^+ \cup \infty$ every time the agent visits a dead-end, meaning that the agent pays this penalty and the search (*trial*) stops. The difference between these criteria is the cost of D. The first criteria would assign a finite penalty, and the second an infinite penalty.

Finite Penalty

In GMDP problems with unavoidable dead-ends and finite penalty D, denoted by fSSPUDE, the agent must pay a finite positive penalty D when encountering a dead-end, and the search stops. However, there are two main problems with this approach. First, the agent does not necessarily knows when it has reached a dead-end. And second, states that are on the path to a dead-end may end having a higher cost that the dead-end itself. As an example consider the a situation in which a state s there is only one action applicable a leading to a dead-end with a probability $1 - \epsilon$, and with probability ϵ to the goal. And the cost of action a is $\epsilon(D + 1)$ In this situation, the expected accumulated cost of s would be $D + \epsilon > D$, hence s would be a worst state than a dead-end.

To avoid these kind of situations, there should be an upper bound to the maximum expected cost of a state. We denote by $J_D^\pi(s)$ the expected limited cost of a state s following policy π:

$$J_D^\pi(s) = \min[D, J^\pi(s)] \tag{7.10}$$

An interesting property of fSSPUDE is that they should be solved by any VI based algorithm because they are also SSP-MDP:

Theorem 5. *f*SSPUDE = SSP-MDP *(Kolobov et al., 2012)*

Proof. Consider that any fSSPUDE can be transformed to another problem with an special action $a_{give-up}$, that leads the agent to a goal state with probability 1 and cost D. Since this action allows to reach the goal from every state, assumption **S1** of SSP-MDP holds. To demonstrate that every SSP-MDP is also an fSSPUDE, we can construct an equivalent fSSPUDE version of an SSP-MDP problem, where $D = J^\pi(s)$. This will ensure that the set of optimal policies in both problems will be the same. □

Value Iteration By Theorem 5, is possible to solve the fSSPUDE using a VI based algorithm. However, the Bellman Equation (Equation 7.8) must be modified to take into account the upper bound, as explained above:

$$V_{t+1}(s) \leftarrow \min\{D, \min_{a \in A}[\mathcal{C}(s, a) + \sum_{s' \in S} \mathcal{P}(s, a, s')V_t(s')]\}, \tag{7.11}$$

initializing $V_0(s)$ $\forall s \notin G$ with arbitrary values, and $V_0(s) = 0$ $\forall s \in G$.

Heuristic search All the properties of the F & R strategy will be maintained if it is used Equation 7.11 instead of Equation 7.8 for updating the values at each step.

Infinite Penalty

In GMDP problems with unavoidable dead-ends and infinite penalty D, denoted by iSSPUDE, the cost of penalty $D = \infty$, meaning that dead-ends are irreparable situations. In this case, all

policies have an infinite expected accumulated cost, and comparing policies based on their expected accumulated cost is not informative, causing all VI and F & R based algorithms to fail to calculate a policy. This leaves us with two different approaches. We could ignore costs and focus on maximizing the probability of reaching the goal, that is we focus on maximizing the probability of reaching the goal from a state s, following a policy π, $P_G^\pi(s)$ (Equation 7.3). Or we could ignore the cost of the histories that do not reach the goal, and focus on finding the policy that minimize the expected cost *over the histories that reach the goal*, that is $V_G^\pi(s)$ (Equation 7.5).

This transforms an iSSPUDE into a multi-objective MDP with two distinct objectives, that is, maximize the probability of reaching the goal and minimizing the expected cost. A multi-objective MDP models problems where competing objective exist and the solutions are *Pareto Sets* of non-dominated policies (Roijers et al., 2013). Remember that a policy π_1 dominates another policy π_2, if for all objectives, π_1 is at least as good as π_2, and at least strictly better in one objective.

Kolobov et al. (2012) define a lexicographic criteria, that evaluates a policy π defining the cost of a state as the ordered pair:

$$J_\infty^\pi(s) = (P_G^\pi(s), V_G^\pi(s)). \tag{7.12}$$

We need a preference operator $\succ$ to indicate which policy is preferable when comparing two different policies. We write $\pi_1 \succ \pi_2$, meaning that π_1 is preferable to π_2, if $J_\infty^{\pi_1}(s) \succ J_\infty^{\pi_2}(s)$, that is, if $P_G^{\pi_1}(s) > P_G^{\pi_2}(s)$, or $P_G^{\pi_1}(s) = P_G^{\pi_2}(s)$ and $V_G^{\pi_1}(s) < V_G^{\pi_2}(s)$. Notice the second condition is conditioned on both policies having the same probability of reaching the goal. Thus, the optimal policy, π^* is the policy that maximizes $J_\infty^\pi(s_0)$ according to the preference operator $\succ$:

$$\pi^* = arg\max_{\succ\pi} J_\infty^\pi(s_0) = arg\max_{\succ\pi}(P_G^\pi(s), V_G^\pi(s)), \tag{7.13}$$

where $max_{\succ\pi}$ is the maximization operator according to the order induced by $\succ$.

This policy can be obtained in two steps, first compute the set of policies $\Pi_{maxprob}$ that maximize the probability to reach the goal, and then select the policy π that minimizes the expected accumulated cost between all policies from $\Pi_{maxprob}$:

$$\Pi_{maxprob} = \{\pi' \text{ s.t. } \pi' = arg\max_\pi P_G^\pi(s)\} \tag{7.14}$$

$$\pi^* = arg \min_{\pi\in\Pi_{maxprob}} V_G^\pi(s)\} \tag{7.15}$$

We can now define formally an SSPUDE as:

Definition 29 (SSP-MDP with unavoidable dead-ends)**.** *An* SSP-MDP *with unavoidable dead-ends, called* SSPUDE *(Kolobov et al., 2012), is a tuple* $\mathcal{M} = \langle S, A, \mathcal{P}, \mathcal{C}, s_0, G, D\rangle$ *where* $S, A, \mathcal{P}, \mathcal{C}, s_0, G$ *are defined as in a* GMDP *(Definition 23) and:*

- $D \in \mathbb{R}^+ \cup \infty$ *is a penalty given to the agent when visiting a dead-end.*

If $D < \infty$*, then* $\mathcal{M}$ *is a finite penalty* SSPUDE *(f*SSPUDE MDP*). The optimal solution to an f*SSPUDE MDP *is a policy that minimizes the expected* ***limited*** *cost from the initial state* s_0*, that is* $\pi^* = arg\min_\pi J_D^\pi(s)$ *(Equation 7.11)*

If $D = \infty$*, then* $\mathcal{M}$ *is an finite penalty* SSPUDE *(i*SSPUDE MDP*). The optimal solution to an i*SSPUDE MDP *is a policy such that* $\pi^* = arg\min_{\pi\in\Pi_{maxprob}} V_G^\pi(s)\}$ *(Equations 7.14 and 7.15).* ◇

Figure 7.5 shows two examples of iSSPUDE MDP, where no proper policy exists. Consider the following policies for the iSSPUDE MDP showed in Figure 7.5 (a): $\pi_1 = \{s_0 : a_0, d_0 : a_2, d_1 : a_3\}$ and $\pi_2 = \{s_0 : a_1, d_0 : a_2, d_1 : a_3\}$. And for the iSSPUDE MDP showed in Figure 7.5 (b): $\pi_1 = \{s_0 : a_0, d_0 : a_2, d_1 : a_3\}$ and $\pi_2 = \{s_0 : a_1, d_0 : a_2, d_1 : a_3\}$. According to the iSSPUDE MDP criteria:

- In Figure 7.5 (a), even if both actions lead the agent to the goal, the agent should apply action a_1 in state s_0 because this action has a probability of reaching the goal of 0.6 instead of applying action a_0 that only has a probability of 0.4.

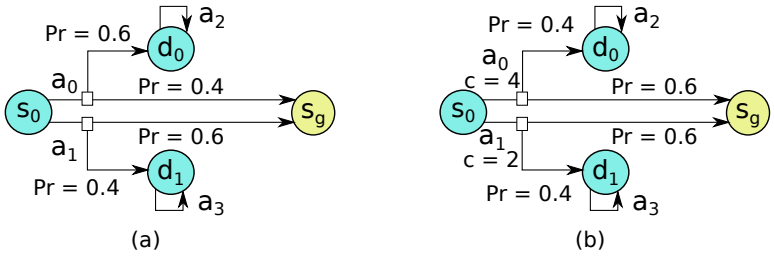


Figure 7.5: *Example of two* iSSPUDE MDP *problems. (a) different action can lead the agent to the goal with different probabilities; and (b) different action can lead the agent to the goal with the same probability but different cost. Unary costs and actions in goal states are omitted for simplicity.*

- In Figure 7.5 (b), both policies π_1 and π_2 have the same probability of reaching the goal. But the agent should apply policy π_2, because this policy has a smaller accumulated cost of the histories that reach the goal.

Equivalence of the Optimization Criteria

Even if the algorithms that solve iSSPUDE MDP are more complicated that those that solve fSSPUDE MDP, both should return the same policy for a given problem $\mathcal{M} = \langle S, A, \mathcal{P}, \mathcal{C}, s_0, G, D \rangle$ if the value of the penalty D is large enough:

Theorem 6. *(Kolobov et al., 2012) For* iSSPUDE *and* fSSPUDE MDP *on the same problem, there exists the smallest finite penalty D_{min} such that for all $D > D_{min}$ the set of optimal policies of the* fSSPUDE MDP *is identical to the optimal policy of* iSSPUDE MDP*.*

Proof. The intuition of the proof is that as D increases, it will prevent the agent of selecting actions that lead it to a dead-end, as it will start increasing the expected value cost of policies with lower probability of reaching the goal than the optimal policy π^*. Consequently any other optimal policy with a higher probability of reaching the goal will be selected. □

However, when the value of the penalty $D < D_{min}$, the policy obtained not only will be sub-optimal in terms of probability but even for a fixed D there may be different policies with the same expected accumulated cost and disparate probabilities of reaching the goal. This is due to the fact that any cost-based criteria is oblivious to the probabilities, and only optimize the probability indirectly, as we will see later.

7.3.4 Maximization of Probability

Another way of treating with the GMDP with dead-ends, is to consider only the probability of reaching the goal, and ignore the costs. This approach, called *Maximum Probability* MDP (MAXPROB MDP) has received a lot of attention recently (Camacho et al., 2016; Kolobov et al., 2011; Steinmetz et al., 2016). The goal of a MAXPROB MDP is to find the policy that maximizes the probability of reaching the goal, from the initial state s_0:

$$P_G(s) = \begin{cases} 1, & \text{if } s \in G; \\ \max\limits_{a \in A} \sum\limits_{s'} \mathcal{P}'(s, a, s') P_G(s'). & \text{otherwise,} \end{cases} \tag{7.16}$$

$$\pi' = arg \max_{\pi} P_G(s_0) \tag{7.17}$$

Kolobov et al. (2011) use the F & R strategy to obtain this policy, by considering only the probabilities and ignoring the costs. But Equation 7.16 can have multiple non-optimal fixed point solutions. For example consider the GMDP showed in Figure 7.6, adapted from Trevizan et al. (2017). This GMDP has the non-optimal fixed point solution to Equation 7.16: $P_G(d_1) = 0$ and $P_G(G) =$

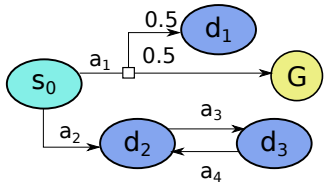


Figure 7.6: *Example* GMDP *with multiple fixed-point solutions. Nodes and solid edges represent, respectively, states and actions. Dead-ends are represented by blue nodes, and goal states by gold nodes.*

$P_G(d_2) = P_G(d_3) = 1$. With this initialization, $P_G(s_0)$ will result in choosing action $a2$ leading the agent to a dead-end.

$$P_G(s_0) = max\{\mathcal{P}'(s_0, a_1, G)P_G(G) + \mathcal{P}'(s_0, a_1, d_1)P_G(d_1), \mathcal{P}'(s_0, a_2, d_2)P_G(d_2)\}$$
$$P_G(s_0) = max\{0.5, 1\}$$
$$P_G(s_0) = 1 \qquad \text{Following action } a_2$$

Thus, one approach to solve Equation 7.16 without finding a non-optimal solution, consist in apply a Value-Iteration algorithm, initializing with 1 all states $g \in G$, and 0 otherwise. Another possible approach is to use an heuristic search algorithm for SSPs such as FRET and FRET-π that are able to remove cycles (Kolobov et al., 2011; Steinmetz et al., 2016), ensuring convergence from any initialization. Or to use linear programming reformulations of the problem as in Trevizan et al. (2017).

Finally, it is interesting to note that the criterion of MAXPROB MDP do not distinguish between policies with the same probability but different costs. Hence the MAXPROB MDP criterion only works in SSP-MDP problems where all proper policies have the same expected accumulated cost.

7.4 Discussion about this chapter

In this chapter we have presented the foundations of probabilistic planning problems, with full or partial observability. We have also shown how to solve SSP problems, using a state-of-the-art approach asynchronous search to solve these problems. However, we have also showed how dead-ends present a major challenge for solvers, and how it is difficult to come up with solutions in the presence of dead-ends. To do so, we have introduced two different types of problems with dead-ends and how to tackle them.

However, these approaches fail to solve the problems, giving up when encountering a dead-end instead of asking human help. Thus, in the next chapter we show how to introduce human help for GMDP problems with dead-ends, and explain different criteria to guarantee to find strong solutions using the minimal human help.

Chapter 8

Goal-Oriented MDPs with Human Help

In Chapter 4, we have shown how to use human help to obtain strong (cyclic) policies for contingent planning problems with dead-ends, where only weak policies exist. We also have explained how to compute the human help actions by exploring the causal graph. In this chapter, we will extend our approach to consider probabilistic planning problems with dead-ends, i.e. SSPADE or SSPUDE problems. As with weak policies in contingent planning problems, SSPUDE problems have also improper policies that do not reach the goal for all states. To this end, we will show how the agent can resort to human help actions in such scenarios, adapting the human help actions explained in previous chapters.

Contrary to contingent or conformant problems, SSPADE or SSPUDE problems are fully observable, hence no observations are needed. However, unlike contingent planning problems, where there exist neither costs nor probabilities and the only criterion is to find strong policies and then minimize the **size** of the policy, in probabilistic planning problems there exists other criteria, as we have seen in the previous chapter, namely minimizing the expected cost or maximizing the probability of reaching the goal. However, it must be noted that in such complex environments, human help actions are partly defined because of the difficulty of associating costs with them. This prevent us from finding a solution by directly minimizing the expected cost.

In this chapter, we will start by describing a generalization of goal-oriented markov decision processes (GMDPs), called *goal-oriented markov decision process with human help* (GMDP-HH), which encompasses markov decision processes with goal and dead-end states, where an agent reasons about human help using artificial actions able to modify any fluent of the problem. These somehow artificial actions enable agents to plan beyond dead-ends, and allow us to introduce human actions into any given GMDP. When these human actions are unknown, we create them from the set of fluents F of the problem, as showed in Chapter 4. The rationale is that during execution some human action will be taken in order to modify the same fluent modified by such an artificial action, but possibly will also modify other fluents. If the agent considers that the human has enough competences to modify all the fluents of the problem, the problem can be seen as an SSP, as we will prove later. However, this large set of fluents can be restricted to account for the limitations of the humans in the environment, that is, to reflect what a human can or cannot modify. In this work, we will assume that there always exists a proper policy from every state when using human help, i.e., GMDP-HH problems are SSP.

Consider the GMDP problem P showed in Figure 8.1 (left). Algorithms that solve SSP-MDPs problems would not find a policy for this problem, because: *(i)* if a_2 is the chosen action for state s_1, the value of state s_1 will never converge, i.e., it will increase to ∞; and *(ii)* if a_3 is the chosen action for s_1 the value of s_1 will be $-\infty$, and s_g is not reached. State d_0 is a *dead-end* and the problem P is not an SSP-MDP since it violates assumptions **S1** and **S2** (Definition 26), and only improper policies exist. Consider now the same problem with human help actions, as showed in Figure 8.1 (right). In this case, there exists a policy in which the agent is able to reach the goal with probability 1 from the initial state, by asking human help.

In order to increase the robustness and autonomy of robots, we assume that these human help

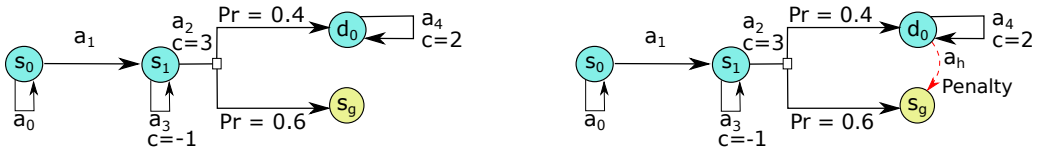


Figure 8.1: *Examples of* GMDP *problems. Left: P with a dead-end (state d_0) and negative cost cycles (in state s_1). Right: The same problem with human help actions (a_h). Unary costs and actions in goal states are omitted for simplicity.*

actions are a scarce resource to be used only if necessary. In this Chapter we will start by formally describing a *goal-oriented markov decision process with human help* (GMDP-HH). To solve these problems, we seek a proper policy (one that reaches the goal with certainty) that minimizes the probability of using human help. We call this criterion **MinHProb** (*minimizing the human-help probability*). While appealing, solutions using this criterion are difficult to obtain, as we will shown shortly, because the corresponding Bellman equation has multiple non-optimal fixed points and heuristics for probability estimation are usually inefficient (Trevizan et al., 2017).

Next, we will consider an alternative class of decision problems, called **GMDPs with a Penalty on Human Help (GMDP-PHH)**, where a finite (positive) penalty is incurred only the first time a human help is used. This can be seen as modeling a situation where requesting the presence of a human is expensive, but once the human is available subsequent calls for human help are cheap. An optimal policy minimizes then the expected cumulative cost (which includes the penalty for using a human help for the first time); we call this criterion **MinPCost**.

The one-time penalty leads to optimal policies that are non-Markovian, because the cost of the human action would depend on the history until that point, not only on the current state. To avoid dealing with such policies, we instead operate over an augmented state space (where states are augmented with a fluent h indicating whether they were reached with the help of human). This allows us to formulate the problem as a standard stochastic shortest path MDP (SSP), and employ any of the state-of-the-art SSP solvers available.

We connect both classes of problems (GMDP-HH and GMDP-PHH) by proving that, for a large enough penalty, the MinPCost criterion finds policies with *minimum probability of using human help*, that is, which are also optimal under the MinHProb criterion. While there is no known strategy for finding a "large enough" penalty, our empirical results show that it is often possible to efficiently find one by linear search (that is, by solving for MinPCost problems with increasingly large penalty values until the optimal policy stabilizes).

However, as we show here, simply minimizing the probability of using human action encourages the robot to maximize the number of human actions whenever a human action is used, since this decreases the cost of robot actions and incurs no additional cost and maximizes the probability of reaching the goal. To alleviate such problem, we propose instead to minimize the frequency of human actions, breaking ties by the expected cumulative cost of robot actions. We show that optimal policies under this criterion can be obtained by optimizing a proxy criterion that assigns a large uniform cost over human actions and then minimizes the expected cumulative cost (of robot and human actions). We call this criterion **MinUCost**. While optimizing for this criterion does not guarantee that human help is used with minimum probability, for a large enough cost of human action, we show that optimal policies according to this criterion minimize the expected number of human actions. This approach has also the advantage of benefiting from the large tool set of SSP MDP solvers.

8.1 Goal-Oriented MDP Augmented with Human Help

As showed in Section 7.3, for domains with no proper policy, the Bellman Equation 7.8 might not converge, and policies might have unbounded expected cumulative cost, hence be incomparable. We have also seen some of the several alternative optimization criteria that have been proposed to

cope with this issue (Kolobov et al., 2012; Teichteil-Königsbuch, 2012; Trevizan et al., 2017), but none of them have addressed the behavior of an agent that meets a dead-end, or have considered human assistance in the process.

Executing such an action requires an explicit request for human help, whose cost is usually difficult to specify. More importantly, the human agent might fail to answer the request, and if he does answer, it might take an unpredictable amount of time to do so (especially, if several different human agents are acting in the environment, or if the robot is new to the environment). Not only the competences of the humans, but also the cost of robot actions that may help the agent, is hard to estimate, since human willingness to engage with robots (or autonomous systems in general) varies according to a large number of hard-to-measure variables (e.g. an human that generally is eager to help the robot might refuse help if an important event, unknown to the robot, is taking place that day). This prevents the use of approaches relying on *mixed-initiative planning* (Côté et al., 2012; Mouaddib et al., 2010), where all costs and competences of the human are explicitly described in the description of the problem.

We propose then, a new model for reasoning about human help called goal-oriented markov decision processes with human help (GMDP-HH), a class of probabilistic planning problems which encompasses markov decision processes with goal and dead-end states, where an agent has a distinguished set of human help actions. In this work we allow the agent (e.g. the robot) to be equipped with a special set of operations A_H called **human help actions** that can modify the environment, but that are *not* under the control of the planning robot.

Let $\mathcal{M} = \langle S, s_0, G, A, \mathcal{P}, \mathcal{C} \rangle$ be a GMDP described in a planning domain description language with fluents F and actions A. We call the tuple $\mathcal{M}_H = \langle S \cup S_H, s_0, G \cup G_H, A \cup A_H, \mathcal{P}_H, \mathcal{C} \cup \mathcal{C}_H \rangle$ a **GMDP augmented with human help** (**GMDP-HH**):

Definition 30 (Goal-Oriented Markov Decision Processes with Human Help (GMDP-HH))**.** *Given a* GMDP $\mathcal{M} = \langle S, s_0, G, A, \mathcal{P}, \mathcal{C} \rangle$*, a Goal-Oriented Markov Decision Processes with Human Help (*GMDP-HH*) is a tuple* $\mathcal{M}_H = \langle S \cup S_H, s_0, G \cup G_H, A \cup A_H, \mathcal{P}_H, \mathcal{C} \cup \mathcal{C}_H \rangle$*, where we introduce for every state* $s \in S$*, a state* $s_h = s \cup \{h\}$ *representing that* s *was reached using human help (and* s *now represents that it was reached without human help);* S*,* s_0*,* G*,* A *and* $\mathcal{C}$ *are defined as in* $\mathcal{M}$ *(Definition 23), and:*

- S_H *is the set of all states reached with human help;*
- $G_H = \{s \cup \{h\} : s \in G\}$ *is the set of goal states reached with human help;*
- A_H *is the set of human actions, where* $A_H(s)$ *denotes the human actions applicable in state* $s \in S \cup S_H$*;*
- $\mathcal{P}_H$ *is* $\mathcal{P}$ *extended to account for human actions; and*
- $\mathcal{C}_H$ *is the cost for human actions, and is unknown.*

◇

We could specify a GMDP-HH in a more concise fashion, using an extended set-theoretic STRIPS language of actions:

Definition 31 (Set-theoretic Goal-Oriented Markov Decision Processes with Human Help (GMDP-HH))**.** *Given a* GMDP-HH*, expressed using an extended* STRIPS *language of actions,* $\mathcal{M}_F = \langle F, I, \mathcal{A}, \mathcal{G} \rangle$*, a* GMDP-HH *is a tuple* $\mathcal{M}_F = \langle F', I', \mathcal{A}', \mathcal{G}' \rangle$ *where:*

- $F' = F \cup \{h\}$*;*
- $I' = I \cup \{\neg h\}$*;*
- $\mathcal{A}' = \mathcal{A} \cup A_H$
- $\mathcal{G}' = \mathcal{G} \cup \{\mathcal{G} \cup h\}$*;*

◇

Human Actions Notice that Definition 30 considers that the agent does not know the costs of human actions. The agent may also be ignorant of the capabilities of the agent, and in this case, the agent must infer the **human help actions** A_H from the set of fluents F of the problem, such that $A_H = \{a_f, a_{\neg f} | f \in F\}$ and:

- $\text{prec}(a_f) = \neg f$; $\text{prec}(a_{\neg f}) = f$;
- $\text{cost}(a_f) = c(a_{\neg f}) = c_H$;
- $\text{eff}(a_f) = \langle 1.0, \{f, h\}, \emptyset \rangle$;
- $\text{eff}(a_{\neg f}) = \langle 1.0, \{h\}, \{f\} \rangle$.

Intuitively, the action a_f causes f to be true, while the action $a_{\neg f}$ causes f to be false, and can either be applied in any state in which f is false or true, respectively, at a uniform cost c_H. Notice that human actions obtained from the domain must contain also the fluent h in its effects. In both cases, we assume that the cost of these actions $\mathcal{C}_H$ is unknown, however a GMDP-HH with a set of actions A_H obtained from the descriptions of the problem has interesting properties:

Theorem 7. *A Goal-Oriented Markov Decision Process $\mathcal{M}$ augmented with **human help actions** A_H obtained from the set of **all** fluents of the problem, is an* SSP MDP.

Proof. Recall that a GMDP is an SSP MDP if there is a proper policy for any state. Since in a GMDP-HH human actions can modify the value of any fluent with probability 1, then any goal state can be reached from any state (with prob. 1). □

Theorem 7 is only valid when the agent computes the set of actions from the set F of all the fluents of the problem. However, when the set of **human help actions** A_H is restricted, there is not such guarantee, and the problem may not be an SSP. We consider in this work only GMDP-HH where there exists **at least one proper policy from every state**, possibly using human actions to achieve goals (i.e., we assume the GMDP-HH encodes a SSP).

Given a GMDP-HH $\mathcal{M}_H$ and a policy π, we can rewrite Equation 7.7 as the sum of the expected cumulative cost considering only the robot actions, $V_R^\pi(s)$, and the expected cumulative cost considering only the human actions, $V_H^\pi(s)$, i.e.:

$$V^\pi(s) = V_R^\pi(s) + V_H^\pi(s), \tag{8.1}$$

where $V_R^\pi(s)$ and $V_H^\pi(s)$ can be computed by the equations:

$$V_R^\pi(s) = \begin{cases} 0, & \text{if } s \in G; \\ \min\limits_{a \in A} \left\{ \mathcal{C}(s, \pi(s)) + \sum\limits_{s' \in S} \mathcal{P}(s, \pi(s), s') V^*(s') \right\} & \text{s.t. } \pi(s) \in A. \end{cases} \tag{8.2}$$

and

$$V_H^\pi(s) = \begin{cases} 0, & \text{if } s \in G; \\ \min\limits_{a \in A} \left\{ \mathcal{C}(s, \pi(s)) + \sum\limits_{s' \in S} \mathcal{P}(s, \pi(s), s') V^*(s') \right\} & \text{s.t. } \pi(s) \in A_H. \end{cases} \tag{8.3}$$

8.1.1 Minimizing the Probability of Human Help (MinHProb)

Given that the human help is (generally) a costly resource, but the cost of human action may be unknown to the agent, we cannot select policies by minimizing expected cumulative cost. An intuitive alternative criterion for solving a GMDP-HH is to find a proper policy that minimizes the probability of using human help. We denote this criterion by **MinHProb**. To this end, we define the **probability of reaching a goal using human help** when executing policy π as:

$$P^{\pi}_{G_H}(s) = \sum_{\sigma} \prod_{i=1}^{|\sigma|} \mathcal{P}(s_i, \pi(s_i), s_{i+1}) \,, \tag{8.4}$$

where the sum is over all histories that start at s and end up in some $s_{|\sigma|} \in G_H$. It is important to remark that in GMDP-HH $P^{\pi}_{G_H}(s) + P^{\pi}_{G} = 1$. The optimal policy under the **minimum human help probability** criterion, denoted as **MinHProb**, is:

$$\pi^*_{\text{MinHProb}} \in \arg\min_{\pi} P^{\pi}_{G_H}(s_0) \text{ subject to } P^{\pi}_{G \cup G_H}(s_0) = 1 \,. \tag{8.5}$$

In words, the optimal policy is a proper policy that minimizes the probability of using human help. The requirement of being a proper policy ($P^{\pi}_{G \cup G_H}(s_0) = 1$) is necessary to avoid improper policies that do not use human help (hence have probability zero of reaching the goal with human help). This criterion has the following interesting properties:

Proposition 5. *If the original* GMDP $\mathcal{M}$ *has a proper policy* π^* *for* s_0*, then* $\pi^* \in \arg\min_{\pi} P^{\pi}_{G_H}(s_0)$*. Conversely, if* $P^{\pi_{MinHProb}}_{G_H}(s_0) = 0$ *then the original* MDP *has at least a proper policy* $\pi_{MinHProb}$ *for* s_0*.*

Proof. Note that $P^{\pi}_{G \cup G_H}(s_0) = P^{\pi}_{G}(s_0) + P^{\pi}_{G_H}(s_0)$. Hence, a proper policy π for s_0 in the original GMDP $\mathcal{M}$ satisfies $P^{\pi}_{G}(s_0) = 1$, which implies that $P^{\pi}_{G_H}(s_0) = 0$. Conversely, any policy π with $P^{\pi}_{G_H}(s_0) = 0$ must satisfy $P^{\pi}_{G}(s_0) = 1$, and hence be proper for s_0 in the original problem. □

According to the proposition above, the MinHProb criterion finds a policy π^*_{MinHProb} that uses human help only if necessary, that is, only when the robot finds itself in a dead-end.

8.1.2 Bellman Equation for MinHProb

One possible approach to solve a GMDP-HH, consists in recasting the problem of finding an optimal policy for MinHProb as the solution of the Stochastic Shortest-Path MDP with Unavoidable Dead-Ends SSPUDE $\mathcal{M}_{\text{sspude}} = \langle S \cup S_H, s_0, G, A \cup A_H, \mathcal{P}', \mathcal{C}' \rangle$, where:

- S denotes the states reached without using human actions, $S_H = \{(s, h) : s \in S\}$ denotes the states reached using human help,
- G denotes the goal states reached without using human help,
- $\mathcal{P}'(s, a, s') = 0$ if $a \in A_H$ for $s' \notin S_H$ else $\mathcal{P}'(s, a, s') = \mathcal{P}(s, a, s')$ for $s' \in S$,
- and $\mathcal{C}'(s, a) = \mathcal{C}(s, a)$ for $a \in A$ and $\mathcal{C}'(s, a) = 0$ for $a \in A_H$.

That is, goals reached using human help become (the only) dead-ends in this SSP MDP, and human actions have no cost. Then, π_{MinHProb} is an optimal fixed-point of the Bellman Equation 8.6 (Steinmetz et al., 2016), where $G_H = \{(s, h) : s \in G\}$ denotes the goal states reached with human help (which in $\mathcal{M}_{\text{sspude}}$ corresponding to the set of absorbing dead-ends).

$$P^*_{G_H}(s) = \begin{cases} 0, & \text{if } s \in G; \\ 1, & \text{if } s \in G_H; \\ \min\limits_{a \in A} \sum\limits_{s'} \mathcal{P}'(s, a, s') P^*(s'). & \text{otherwise,} \end{cases} \tag{8.6}$$

where $G_H = \{(s, 1) : s \in G\}$ denotes the goal states reached with human help (which in $\mathcal{M}_{\text{sspude}}$ corresponding to the set of absorbing dead-ends).

However, while solutions that minimize the probability of human help are fixed point solutions of Equation 8.6, it must also be noted that not all fixed points of Equation 8.6 correspond to optimal policies, as we will show shortly.

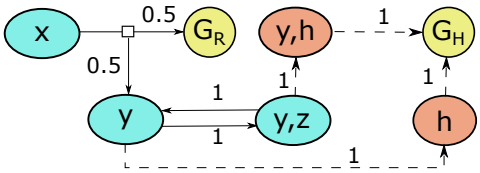


Figure 8.2: GMDP-HH *with fluents* $F = \{x, y, z\}$ *and multiple fixed-point solutions. Nodes, solid edges and dotted edges represent, respectively, states, agent actions and human help actions. The numbers close to edges denote the probability of the respective transition.*

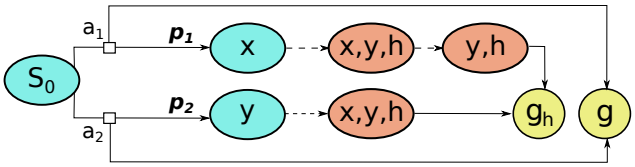


Figure 8.3: *Illustration of a* GMDP-HH *with fluents* $F = \{x, y, z\}$ *and one goal state. Two actions are applicable at* s_0*, resulting in four different histories starting at* s_0*.*

This approach, however, would not work, because one way to minimize $P^*_{G_H}(s)$ is to avoid *any* goals as much as possible, i.e., some fixed-points of Equation 8.6, do not correspond to proper policies. In fact, it is well-known that not every fixed-point of the Equation (8.6) corresponds to a proper policy (Steinmetz et al., 2016; Trevizan et al., 2017). For instance, in the GMDP in Figure 8.2 any solution $P^*_{G_H}(s_1, 0) = P^*_{G_H}(s_2, 0) < 1$ is a non-optimal fixed point that leads to an improper policy (as it never leaves the cycle between states $\{y\}$ and $\{y, z\}$).

A consequence of having multiple fixed-points is that not every initialization of $P^*_{G_H}(s)$ is guaranteed to converge to an optimal solution for Value Iteration algorithms. In particular, admissible heuristics for $P^*_{G_H}$ do not ensure convergence and hence cannot be used, for example assigning a value of 1 to the entire region S_H, leads to very slow convergence (in our experiments, this approach was ineffective in finding optimal solutions even in very small problems). Even considering an initialization of 0 for all states $g \in G$, and of 1 for all the others, this can be misleading. For example, consider Figure 8.2, and an optimal policy according to Equation 8.6 that prescribes in state y the action leading to state y, z, and from state y, z an action leading to state y, that is, a cycle. The probability of reaching the goal with human help of this policy is also 1, however this is not true if the agent does not leave the cycle, because it will never reach the goal even with human help. Situations like this can be avoided if we consider every human action to have a positive but unknown cost ϵ, and we select as the optimal policy for MinHProb, the policy with the minimal expected cost among all policies with the minimal probability of reaching the goal with human help, i.e., solving this problems in two steps.

Another possible solution is to adapt algorithms for SSPs such as FRET and FRET-π that find and remove problematic cycles (Kolobov et al., 2011; Steinmetz et al., 2016), ensuring convergence from any initialization. Or to use linear programming reformulations of the problem as in Trevizan et al. (2017).

Finally, another issue with the MinHProb criterion is that two proper optimal policies might achieve the same probability $P^{\pi}_{G_H}(s_0)$ while executing a very different number of human actions. For example, consider the GMDP-HH in Figure 8.3, and assume that $p_1 = p_2 = p$. Then, selecting either action at s_0 leads to an optimal policy π^*_{MinHProb} with $P^{\pi^*_{\text{MinHProb}}}_{G_H}(s_0) = p$, while executing a different number of human actions (and obtaining different cumulative costs). Situations like these can be remedied by additionally minimizing the expected cumulative cost among policies π^*_{MinHProb}. We could devise a naive solution that solves this in two steps: (1) first computing policies π^*_{MinHProb}; and (2) selecting the minimum cost among these. We denote this approach by **MinHProb-MinUCost**. As we will see in the experiments, this is a highly inefficient approach and inherits the pitfalls of computing π^*_{MinHProb}. We show in the next Section how this two-step criterion can be more efficiently computed using a surrogate criterion that introduces a finite penalty on the

first time a human action is used.

8.2 Goal-Oriented MDP with a Penalty on Human Help

An alternative criterion to find a policy that minimizes human help is to minimize the expected cumulative cost while severely penalizing any history that uses a human help. We denote this criterion by **MinPCost**. Intuitively, this criterion assumes that the cost of human help is amortized if used repeatedly. This is a realistic scenario when there is a high cost of requesting human presence, but a small cost for actually using human help. Thus, we define the **Goal-Oriented MDP with a Penalty on Human Help (GMDP-PHH)** as the tuple $M_{HP} = \langle S \cup S_H, s_0, G \cup G_H, A \cup A_H, \mathcal{P}, \mathcal{C}, D_H \rangle$, where all terms are defined as in a GMDP-HH, and $D_H > 0$ is a finite value denoting the penalty incurred the first time a human action is used.

Solving a GMDP-PHH is akin to solving GMDPs with a give-up action that takes the agent from any state directly into a goal state and incurs a (usually large) finite penalty (Kolobov et al., 2012). Conceptually however a GMPDP-PHH differs from a GMPD with a give-up action (a.k.a. fSPUDE) since in the former the agent resumes planning after paying the penalty D_H.

We can solve a GMDP-PHH efficiently by using any off-the-shelf SSP solver by modifying the cost function $\mathcal{C}(s, a)$ so that it returns $c_H + D_H$ if $s \in S$ and $a \in A_H$, and otherwise remains unchanged. Importantly, setting the value of D_H large enough ensures that the optimal policy π^* minimizes $P^{\pi}_{G_H}(s_0)$, i.e., $P^{\pi^*}_{G_H}(s_0) = P^{\pi^*_{\text{MinHProb}}}_{G_H}(s_0)$, while minimizing expected cumulative cost and satisfying $P^{\pi}_{G \cup G_H}(s_0) = 1$:

Theorem 8. *There exists a value D_{hmin} such that for all $D_H > D_{hmin}$ minimizing the expected cumulative cost with a first human help penalty also minimizes the probability of reaching the goal using human actions.*

Proof. The intuition behind this proof is as follows. For a GMDP-HH, minimizing the expected cumulative cost with a penalty D_H for using human help may obtain a proper policy π' that uses human help with a higher probability $P^{\pi'}_{G_H}(s_0) > P^{\pi^*}_{G_H}(s_0)$, because the expected cost of the histories that use human help is lower.

However for a sufficiently large penalty D_H the increased expected cumulative cost of the policy π' that uses more human help cannot be outweighed by the lower expected cost of the history. Using such penalty, the optimal policy π^* that minimizes the expected cumulative cost must also minimize the probability of using human help.

For a policy π that solves a GMDP-HH, let v be the expected cumulative cost that π incur before reaching a goal state. Let us remember that GMDP-HH are SSP and there exists a proper policy for every state, possibly using human help. Note that using human help following policy π results in a penalty D_H added to the expected cumulative cost, multiplied by the probability of using human help, hence according to 7.7, $v = V^{\pi} + P^{\pi}_{G_H} \cdot D_H$. It is easy to see that v is the results of applying the *Penalty on the First Human Action criterion* to the bellman equation of Equation 7.7.

We are ready to show that for any penalty D_H above a certain finite threshold D_{hmin}, there is no (sub optimal) policy π' with a probability $P^{\pi'}_{G_H}(s_0) > P^{\pi^*}_{G_H}(s_0)$ as good as the optimal policy π in terms of expected cumulative cost. And that every optimal policy in terms of expected cumulative cost is also a policy with the minimal probability of using human help $P^{\pi^*}_{G_H}(s_0)$. Let us consider two policies π_1 and π_2. Consider that π_1 is a sub-optimal policy with a probability of using human help $P^{\pi_1}_{G_H} = P^{\pi_2}_{G_H} + \epsilon$, where ϵ is the minimal difference between the probability of the optimal policy $P^{\pi^*}_{G_H}$ and the probability of any sub-optimal policy $P^{\pi'}_{G_H}(s_0)$. All sub-optimal policies have a probability $P^{\pi'}_{G_H}(s_0)$ s.t. $P^{\pi'}_{G_H}(s_0) - P^{\pi^*}_{G_H} \geq 0$.

$$v_1 = V^{\pi_1} + P^{\pi_1}_{G_H} \cdot D_H$$
$$v_2 = V^{\pi_2} + P^{\pi_2}_{G_H} \cdot D_H$$

Let us suppose that v_1, the expected cumulative cost of π_1 is at least as good as $v_2(s)$, the expected cumulative cost of π_2. Then:

$$\begin{aligned}
&0 \leq v_2 - v_1 && \text{(Replacing } v_1, v_2\text{)}\\
&0 \leq V^{\pi_2} + P^{\pi_2}_{G_H} \cdot D_H - V^{\pi_1} - P^{\pi_1}_{G_H} \cdot D_H\\
&0 \leq V^{\pi_2} - V^{\pi_1} + P^{\pi_2}_{G_H} \cdot D_H - P^{\pi_1}_{G_H} \cdot D_H\\
&0 \leq (V^{\pi_2} - V^{\pi_1}) + (P^{\pi_2}_{G_H} - P^{\pi_1}_{G_H}) \cdot D_H
\end{aligned}$$

Let us consider a value of $D_{\text{hmin}} = \frac{V^{\pi_2} - V^{\pi_1}}{\epsilon}$. Unless all policies of the GMDP-HH problem are optimal, in which case the theorem holds vacuously, $\epsilon > 0$, hence this value D_{hmin} is finite. We call D_{hmin} the value of the penalty that make the subtraction above equal to 0:

$$\begin{aligned}
&0 \leq (V^{\pi_2} - V^{\pi_1}) + (P^{\pi_2}_{G_H} - P^{\pi_1}_{G_H})(\frac{V^{\pi_2} - V^{\pi_1}}{\epsilon}) && \text{(substituting} D_H\text{)}\\
&0 \leq (V^{\pi_2} - V^{\pi_1})(1 + \frac{P^{\pi_2}_{G_H} - P^{\pi_1}_{G_H}}{\epsilon})\\
&0 \leq (V^{\pi_2} - V^{\pi_1})(1 + \frac{P^{\pi_2}_{G_H} - P^{\pi_2}_{G_H} - \epsilon}{\epsilon}) && \text{(substituting } P^{\pi_1}_{G_H}\text{)}\\
&0 \leq (V^{\pi_2} - V^{\pi_1})(1 + \frac{-\epsilon}{\epsilon})\\
&0 \leq (V^{\pi_2} - V^{\pi_1})(1 - 1)
\end{aligned}$$

Now, if we consider a $D_H > D_{\text{hmin}}$, for example $D_H = D_{\text{hmin}} + c$, where $D_{\text{hmin}} = \frac{V^{\pi_2} - V^{\pi_1}}{\epsilon}$ and c is a minimum positive value (integer o real),

$$\begin{aligned}
&0 \leq (V^{\pi_2} - V^{\pi_1}) + (P^{\pi_2}_{G_H} - P^{\pi_1}_{G_H})(\frac{V^{\pi_2} - V^{\pi_1}}{\epsilon} + c) && \text{(substituting} D_H\text{)}\\
&0 \leq (V^{\pi_2} - V^{\pi_1}) + (P^{\pi_2}_{G_H} - P^{\pi_2}_{G_H} - \epsilon)(\frac{V^{\pi_2} - V^{\pi_1}}{\epsilon} + c) && \text{(substituting } P^{G_H}_{\pi_1}\text{)}\\
&0 \leq (V^{\pi_2} - V^{\pi_1}) + (-\epsilon)(\frac{V^{\pi_2} - V^{\pi_1}}{\epsilon} + c)\\
&0 \leq (V^{\pi_2} - V^{\pi_1}) + (-\epsilon)(\frac{V^{\pi_2} - V^{\pi_1} + c\epsilon}{\epsilon})\\
&0 \leq V^{\pi_2} - V^{\pi_1} - V^{\pi_2} + V^{\pi_1} - c\epsilon\\
&0 \leq -c\epsilon
\end{aligned}$$

which contradicts our assumption that the expected cumulative cost of π_1 is at least as good as the cumulative cost of π_2. □

Since MinPCost minimizes the probability of using human help, and it also minimizes the

expected cost, following Theorem 8 we enunciate the equivalence of criteria between MinPCost and MinHProb:

Corollary 8.1. *There exists a value $D_{MinHProb}$ such that for all $D_H > D_{MinHProb}$ a MinPCost policy $\pi_{MinPCost}$ using D_H is also a MinHProb policy. Additionally, $\pi_{MinPCost}$ minimizes the unpenalized expected cumulative cost among all MinHProb policies (i.e., it optimizes the two-step criterion),*

Proof. Given a policy π, we can decompose $V^\pi(s)$ as the sum of expected cumulative costs of robot actions, human actions and the one-time penalty:

$$V^\pi(s) = V_R^\pi(s) + V_H^\pi(s) + P_{G_H}^\pi(s) \cdot D_H, \tag{8.7}$$

where $P_{G_H}^\pi(s)$ is given by Equation 8.4. For large enough D_H, a policy that uses a human help action in a given state has a higher expected cumulative cost than a policy that differs only by the choice of agent action in that same state. Hence, an optimal policy will use a human action only if no agent action can lead the agent out of a dead-end. The same argument shows that MinPCost breaks ties by selecting a policy that minimizes the expected cost of robot and human actions, thus satisfying the lexicographic criterion. □

According to Corollary 8.1, for large enough D_H the optimal policy $\pi_{MinPCost}$ under MinPCost also optimizes MinHProb while minimizing the not penalized expected cumulative cost, that is, $P_{G_H}^{\pi_{\text{MinPCost}}}(s_0) = P_{G_H}^{\pi_{\text{MinHProb}}}(s_0)$ and π_{MinPCost} minimizes $V_R^\pi(s) + V_H^\pi(s)$. However, there is no known procedure for finding the value D_{MinHProb} or even for verifying if a given value satisfies the condition on the theorem. In our experiments we observed that by guessing a sufficiently large value D_H and verifying whether increasing this value changes the optimal policy provides an effective means for finding D_{MinHProb} in practice.

8.3 Minimizing the Frequency of Human Actions

The MinHProb criterion (or the MinPCost for sufficiently large penalty) minimizes the probability of using human help while maximizing the probability of reaching the goal. However, the same criterion tends to maximize the probability of reusing human help, since this leads to smaller costs for the robot and higher probability of reaching the goal without altering the probability of human help. To see this consider the GMDP-HH in Figure 8.3, and assume that $p_1 = p_2 = p$, and that agent actions have unit cost. Selecting either action at s_0 leads to an optimal policy π_{MinHProb} with $P_{G_H}^{\pi_{\text{MinHProb}}}(s_0) = p$, while executing a different number of human actions (and obtaining different cumulative costs). In essence, the issue is that MinHProb (or other equivalent criteria) does not minimize the *frequency* with which human help is used.

We thus propose a new criterion, called **MinHFreq**, that minimizes the expected number of human actions; breaking ties by minimizing the expected cost of the actions of the agent. We define the **expected number of human actions** from a state s as the number of human actions weighted by the probability of applying this human actions:

$$N_H^\pi(s) = \sum_\sigma |i : \pi(s_i) \in A_H| \prod_{i=1}^{|\sigma|} \mathcal{P}(s_i, \pi(s_i), s_{i+1}) \tag{8.8}$$

where the sum is over all histories starting at s, and similarly, the expected number of agent actions:

$$N_H^\pi(s) = \sum_\sigma |i : \pi(s_i) \in A| \prod_{i=1}^{|\sigma|} \mathcal{P}(s_i, \pi(s_i), s_{i+1}). \tag{8.9}$$

Thus, the optimal policy under the criterion that **minimizes the expected number of human actions MinHFreq**, is:

$$\pi^*_{\text{MinHFreq}} \in \arg\min_{\pi} N^{\pi}_{H}(s_0) \text{ subject to } P^{\pi}_{G \cup G_H}(s_0) = 1\,. \tag{8.10}$$

In words, the optimal policy under this criterion is a **proper** policy that minimizes the expected number of using human help. The requirement of being a proper policy ($P^{\pi}_{G \cup G_H}(s_0) = 1$) is necessary to avoid improper policies that do not use human help (hence have $N^{\pi}_{H}(s_0) = 0$).

Finding the MinHFreq policy. Directly finding the MinHFreq policy is difficult since it requires a two-pass solution that first finds all policies that minimize N^{π}_{H} then selects the one(s) that minimize the expected action cost from the initial state. It is also challenging for current solvers, since typical heuristics are poorly informative when minimized expected number of human actions (which for most histories is a small number).

In order to cope with these shortcomings, we propose optimizing a proxy criterion, i.e., a criterion that is easier to compute and computes the same policies, that we call **MinUCost** (minimum uniform human action cost) that minimizes the expected cumulative cost assuming a modified cost function that assigns a uniform cost $c_H > 0$ for any human action. We can then, use any efficient off-the-shelf heuristic solver such as FIND-AND-REVISE algorithms (Bonet and Geffner, 2003), LRTDP (Bonet, 2003) and iLAO* (Hansen and Zilberstein, 2001).

While this criterion enables efficient solutions for a fixed c_H, it does not guarantee that a MinHProb or MinHFreq policy is selected. The following results shows that for large enough c_H, proper policies that do not use human actions will be preferred over improper policies whenever they exist:

Theorem 9. *If there exists two different policies π_1 and π_2, such that $N^{\pi_1}_{H}(s) > N^{\pi_2}_{H}(s)$, then exists a sufficiently large cost for the human actions c^{max}_{H}, such that for any human action cost $c_H > c^{max}_{H}$ the policy π_2 has smaller expected cumulative cost than policy π_1 (i.e., $V^{\pi_1}(s_0) < V^{\pi_2}(s_0)$).*

Proof. The intuition of this proof is as follows. For a GMDP-HH, minimizing the expected cumulative cost considering human costs c_H will obtain a policy π' using more human help while minimizing the expected cumulative cost $N^{\pi'}_{H}(s) > N^{\pi^*}_{H}(s)$, because the expected cost of the histories that use human help is lower, or because the histories using more human help actions have a lower expected number of them $N^{\pi'}_{H}$. However, as this cost c_H grows, the increased expected cumulative cost of the policy π' that uses more human help cannot be outweighed by the lower expected cost of the robot actions, if there exists other policy with a smaller expected number of human actions. When c_H is sufficiently large, the agent that minimizes the expected cumulative cost will select the optimal policy π^* with the minimal expected number of human actions. Consider two policies π_1 and π_2, such that $N^{\pi_1}_{H}(s) > N^{\pi_1}_{H}(s)$, and a state s with a probability $0 < P^{\pi}_{G}(s) < 1$:

$$V^{\pi_1}(s) \geq N^{\pi_1}_{H}(s) c_H$$

V^{π_1} is at least equal to the expected number of human actions multiplied by the cost of these actions. This bound is obtained by discarding the cost of every other agent action of policy π_1.

$$V^{\pi_2}(s) \leq N^{\pi_2}_{H}(s) c_H + N^{\pi_2}_{\overline{H}}(s) \max \mathcal{C}$$

For V^{π_2} we can obtain a higher bound if we multiply the expected number of human actions of policy π_2 and add it to the expected number of agent actions multiplied by the maximum cost of the agent actions. For clarity we denote this maximal cost $\max \mathcal{C}$ as c_m.

$$
\begin{aligned}
0 &\leq V^{\pi_2}(s) - V^{\pi_1}(s) && \text{(Replacing } V^{\pi_1}(s), V^{\pi_2}(s)\text{)}\\
0 &\leq N_H^{\pi_2}(s)c_H + N_{\overline{H}}^{\pi_2}(s)c_m - N_H^{\pi_1}(s)c_H\\
0 &\leq N_{\overline{H}}^{\pi_2}(s)c_m + c_H(N_H^{\pi_2}(s) - N_H^{\pi_1}(s))\\
-N_{\overline{H}}^{\pi_2}(s)c_m &\leq c_H(N_H^{\pi_2}(s) - N_H^{\pi_1}(s))\\
\frac{-N_{\overline{H}}^{\pi_2}(s)c_m}{N_H^{\pi_2}(s) - N_H^{\pi_1}(s)} &\leq c_H
\end{aligned}
$$

Since $N_H^{\pi_1}(s) > N_H^{\pi_2}(s)$, the fraction $\frac{-N_{\overline{H}}^{\pi_2}(s)c_m}{N_H^{\pi_2}(s)-N_H^{\pi_1}(s)}$ is positive, hence c_H is a finite positive number. We denote this cost as $c_H^{\max}$, the value of the penalty that makes the subtraction $V^{\pi_2}(s) - V^{\pi_1}(s)$ equal to 0. Now, if we consider a cost for the human actions $c_H > c_H^{\max}$, for example $c_H = c_H^{\max} + \epsilon$, where $c_H^{\max} = \frac{-N_{\overline{H}}^{\pi_2}(s)c_m}{N_H^{\pi_2}(s)-N_H^{\pi_1}(s)} + c$ and ϵ is a minimum positive value (integer o real):

$$
\begin{aligned}
&0 \leq V^{\pi_2}(s) - V^{\pi_1}(s) && \text{(Replacing } V^{\pi_1}(s), V^{\pi_2}(s)\text{)}\\
&0 \leq N_H^{\pi_2}(s)c_H + N_{\overline{H}}^{\pi_2}(s)c_m - N_H^{\pi_1}(s)c_H\\
&0 \leq N_{\overline{H}}^{\pi_2}(s)c_m + c_H(N_H^{\pi_2}(s) - N_H^{\pi_1}(s))\\
&0 \leq N_{\overline{H}}^{\pi_2}(s)c_m + (\frac{-N_{\overline{H}}^{\pi_2}(s)c_m}{N_H^{\pi_2}(s) - N_H^{\pi_1}(s)} + \epsilon)(N_H^{\pi_2}(s) - N_H^{\pi_1}(s))\\
&0 \leq N_{\overline{H}}^{\pi_2}(s)c_m + (\frac{-N_{\overline{H}}^{\pi_2}(s)c_m + \epsilon(N_H^{\pi_2}(s) - N_H^{\pi_1}(s))}{N_H^{\pi_2}(s) - N_H^{\pi_1}(s)})(N_H^{\pi_2}(s) - N_H^{\pi_1}(s))\\
&0 \leq N_{\overline{H}}^{\pi_2}(s)c_m - N_{\overline{H}}^{\pi_2}(s)c_m + \epsilon(N_H^{\pi_2}(s) - N_H^{\pi_1}(s))\\
&0 \leq \epsilon(N_{\pi_2}^H(s) - N_{\pi_1}^H(s)),
\end{aligned}
$$

this is a contradiction since $\epsilon(N_H^{\pi_2}(s) - N_H^{\pi_1}(s))$ will always be negative, which contradicts our assumption that the expected cumulative cost of π_1 is at least as good as the cumulative cost of π_2. □

As an example, consider Figure 8.3. Consider two policies π_1 and π_2, that prescribe actions a_1 and a_2 for the initial state s_0 respectively. The expected number of human actions of s_0, following policy π_1 $N_H^{\pi_1}(s_0) = (1-p_1)\cdot 2\cdot c_H$, and for π_2 $N_H^{\pi_2}(s_0) = (1-p_2)\cdot c_H$ (Eq. 8.8). According to Theorem 9, for a high cost c_H the policy selected will be the one that minimizes the expected number of human actions, hence π_1 will be selected only if $(1-p_1)\cdot 2\cdot c_H < (1-p_2)\cdot c_H$ and π_2 if $(1-p_1)\cdot 2\cdot c_H > (1-p_2)\cdot c_H$.

Note that this result is slightly weaker than Theorem 8, since it cannot be applied in situations with equal expected number of human actions. For example, in the GMDP-HH in Figure 8.3, if we assume that $p_2 = 0, 1-p_2 = 1$ and $p_1 = 0.5, 1-p_1 = 0.5$, and that $C_h = 1$ then $N_H^{\pi_1}(s) = N_H^{\pi_2}(s) = 1$, and the optimal MinUCost depends on the cost of robot actions.

As with MinHProb, the MinUCost uses human actions only in dead-ends:

Corollary 9.1. *There exists a value c_H^{max} such that for any $c_H > c_H^{max}$ any optimal policy prescribes a human action if and only if* $\max_\pi P_\pi^G(s) = 0$.

Proof. It is easy to see that as long as $P_G^\pi(s) = 1$, the expected number of human actions $N_H^\pi(s_0) = 0$. Thus any policy that does not use human actions may be selected. Consider two policies π_1 and π_2, and a state s with a probability $0 < P_G^\pi(s) < 1$, and $\pi_1(s) \in A_H$ and $\pi_2(s) \notin A_H$. Consider that in the state s, the expected cumulative cost of π_1, V^{π_1} is at least as good as the expected cumulative cost of π_2 V^{π_2}:

$$V^{\pi_2} - V^{\pi_1} \geq 0$$

Since $\pi_1(s) \in A_H$, $V^{\pi_1} \geq c_H$, and it is sufficient to set a value V^{π_2} for the cost c_H to make $V^{\pi_2} - V^{\pi_1} = 0$. We call that value $c_H^{\max}$, and for any value $c_H > c_H^{\max}$ $V^{\pi_2} - V^{\pi_1} \leq 0$, which contradicts the assumption that the expected cumulative cost of π_1, V^{π_1} is at least as good as the expected cumulative cost of π_2 V^{π_2}. □

In conclusion, even though using large values of c_H discourages unnecessary use of human actions, the optimal policy according to MinUCost might not minimize the probability of reaching the goal using human help, as showed above, but it will obtain a policy that minimizes the frequency of using human help.

8.4 Discussion

In this chapter we have introduced models to deal with dead-ends in GMDP using human actions. These models use different algorithms that optimize a given optimality criteria to obtain different solutions. These optimality criteria could be classified in different types, and there exist a similarity between our criteria and those showed in Section 7.3: (1) lexicographic criteria, for example iSSPUDE and the **MinHProb-MinUCost** criterion; (2) ignore costs and focus on probabilities, like MAXPROB MDP and **MinHProb**; (3) ignore probabilities and focus on modified costs, like **MinUCost**; and (4) using different parameters, like fSSPUDE and **MinPCost**.

The first class, represent an agent that wants to focus first in reaching the goal without using human help except when necessary, and second in minimizing the expected cumulative cost (**MinHProb-MinUCost**). Algorithms proposed to compute solutions using these criteria must work by approximating two separate functions, which is computationally expensive.

The second category represents an agent that focus exclusively on minimizing one objective, ignoring the rest. For example, the **MinHProb** criteria that minimizes the use of human help, regardless of the cost.

The third category is a novelty of this work, and consist also on minimizing one objective, in this case the expected cumulative cost while indirectly minimizing other objective. This criterion could use the state-of-the-art planners to compute the optimal policies knowing that minimizing the expected cumulative cost is guaranteed to minimize the expected number of using human actions. However, this is not always true, and it depends on the action costs be high enough.

The last category uses predetermined parameters to compute the optimal policy minimizing more than one objective. For example, the penalty D_H in **MinPCost** compute a policy that minimizes the probability of using human help. However, this is not always true for all values of this parameters, for example with lower penalties the agent may end using human help more than necessary.

In general, knowing the dead-ends could improve the performance of all of this approaches. For example, when updating the values in a step of a VI, in a dead-end state the agent should know that the probability of reaching the goal is zero, or the probability of using human help is 1. Dead-ends could then be used to create the set of human actions, for example limiting the fluents the human can modify only to those that cause a dead-end.

We have also shown in Chapter 4.4, how to compile knowledge from the causal graph to obtain more informative actions, that even if they may not correspond to a dead-end, they could approximate it close enough. The criteria defined here then would minimize the use of such actions, using them only in the most urgent case. The human actions in the optimal policy then should explain why the agent will fail, hence explaining the dead-end.

Chapter 9

Experiments in Probabilistic Planning with Human Help

The objective of the experiments in this section is to evaluate our proposed solutions for Probabilistic Planning problems using the optimization criteria presented in Sections 8.3 and 8.2. We show experiments for the *MinUCost* and *MinPCost* criteria, increasing the cost of the human actions C_H and the penalty for using human help D_H, respectively. Experiments with *MinHProb* criteria are only shown for a small instance since the Value Iteration based algorithm to solve Equation 8.6 is highly inefficient. However, according to Theorem 8, the optimal policy under *MinPCost* for large enough D_H is also the optimal policy for the *MinHProb-MinUCost* criterion.

As in Chapter 5, we start this Chapter by describing the different domains used to test our ideas, explaining their statistics and why they are interesting for our tests. And in the second part we show how to obtain proper policies for the Probabilistic Problems, increasing the cost of the human actions c_H and the penalty for using human help D_H and using different criteria, the *MinUCost* and *MinPCost* criteria respectively.

9.1 Domains

For the Probabilistic problems, we used the following domains in our experiments: a probabilistic version of Doors (*doors*), Navigation (*navigation*) and Triangle Tire World (*tireworld*). Doors and Navigation correspond to SSPUDE domains, and Triangle Tire World is a SSPADE domain. The domain Doors (*doors*) correspond to a probabilistic version of the same domain explained in Section 5.1.6. In the contingent version, the agent was able to *observe* if the key was at a location. But in the probabilistic version, there is a 50% of probability that the key is not in the room where the agent searches for it. This makes this domain a SSPUDE since there is always the possibility that the keys are not in the room.

9.1.1 Navigation

The *Navigation* domain was described originally in RDDL language (*Relational Dynamic Influence Diagram Language*) (Sanner, 2010) for the International Probabilistic Planning Competition of 2011 (Sanner and Yoon, 2011). In this domain, a robot starting at an initial position I must reach a goal location G. The robot is able to move in any of the 4 directions (*north*, *west*, *south* and *east*), but when moving *north* there is a chance that the robot breaks, that is, a *dead-end* because the robot to be unable to continue. However, this probability of breaking the robot breaking diminishes as the agent moves towards the west, to a minimum of 0.10.

An interesting thing in this domain, is that there is no proper policy from the initial state, because to reach the goal location, the robot must move to the *north* at some moment, hence this problem is a SSPUDE. Notice in Figure 9.1 how there is always a probability to break the robot

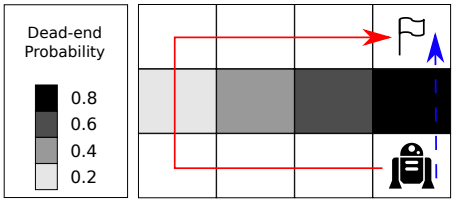


Figure 9.1: *Example of an instance of Navigation problem. The shades of gray indicate the probability of breaking the robot.*

Instance	Size	$\|S\|$	Dead-Ends	Actions
navigation-6	186	187	1	744
navigation-7	216	217	1	864
navigation-8	246	247	1	984
navigation-9	276	277	1	1104
navigation-10	306	307	1	1224

Table 9.1: *Statistics for 5 instances of Navigation domain. Size indicates the size of the map, Total States indicates the total number of possible states in the problem, Dead-ends the possible number of dead-ends and Actions the total number of actions.*

when moving to the north. The variables of this problem encode only the location of the robot in the map.

Table 9.1 shows the statistics of an instance of the Navigation problem with 100 columns and 3 rows. Because there is only one variable, the *dead-end* in this domain is easy to detect.

9.1.2 Triangle Tire World

The last domain we tested, *Triangle Tireworld* is a domain that was created for the International Probabilistic Planning Competition of 2004[1]. In this domain, an autonomous car must reach a goal location G from an initial location I, while moving in different roads. Every time the car moves, there is a chance that the tire will flat, stopping the car and making it unable to continue. Some locations contain a spare tire, and in these locations it is possible to change the tire and continue. The *dead-ends* of this domain consist in being in a location without a spare tire. While there exists always a probability of flatting the tire, there is always a safe path, where the agent can change the tires. Figure 9.2 shows two instances of this domain. Notice that there is a path from the initial location to the goal location, visiting only locations with spare tires (black circles).

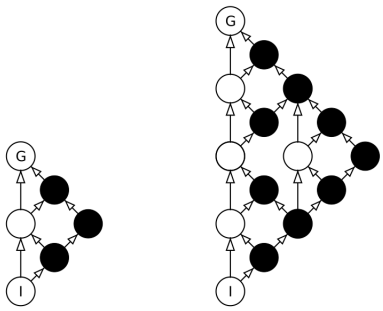


Figure 9.2: *Example of two instances of Triangle Tireworld. I stands for the initial location, and G stands for the goal location. Black circles indicate locations with a spare tire, and white circles locations without.*

[1] www.cs.rutgers.edu/ mlittman/topics/ipc04-pt/

Instance	Size	Spares	$\|S\|$	Dead-Ends	Actions
Tireworld-4	81	28	21743271936	81	162
Tireworld-5	121	40	133040906960896	121	242
Tireworld-6	169	54	3044433348102455296	169	338

Table 9.2: *Statistics for 5 instances of Triangle Tireworld domain. Size indicates the size of the map, Spares indicates the number of spare wheels in the map, Total States indicates the total number of possible states in the problem, Dead-ends the possible number of dead-ends and Actions the total number of actions.*

Table 9.2 shows the statistics for this domain. The large number of states contains some impossible states, mainly because when the car has a flat tire, it is unable to move in any direction. The number of dead-ends is small because it consists in the car in a location with no spare tire.

9.2 Goal-Oriented MDP with a Penalty on Human Help

The objective of these experiments is to evaluate our proposed solutions for GMDP-HHP problems using the optimization criteria presented in Section 8. In this section, we show experiments for the *MinPCost* criteria for the three different domains *Doors*, *Tireworld* and *Navigation*, increasing the cost of the penalty for using human help D_H, respectively.

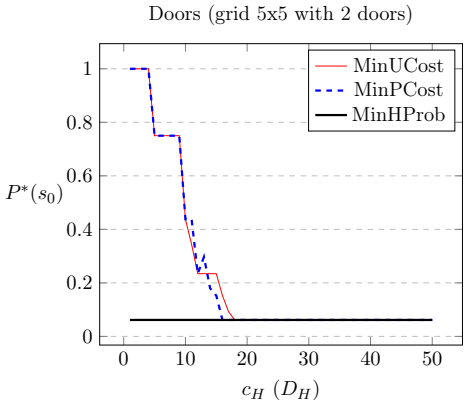


Figure 9.3: *Optimal expected probability of using Human Help from s_0, increasing the cost c_H and the penalty D_H for the instance Doors-5.*

Experiments with *MinHProb* criteria are showed only for a small instance since the Value Iteration based algorithm to solve Equation 8.6 is highly inefficient. However, according to Theorem 8, the optimal policy under *MinPCost* for large enough D_H is also the optimal policy for the *MinHProb-MinUCost* criterion.

To solve GMDP-HHs under *MinUCost*, we used the LRTDP algorithm (Bonet, 2003) implemented on the mGPT Framework (Bonet and Geffner, 2005b). To solve the GMDP-HH under *MinPCost* we used a modified version of LRTDP that includes a function to verify if a state s satisfies the fluent h, i.e. $h \in s$.

All experiments were performed in a Linux machine with a 2.4 GHz processor and 213GB RAM of memory, with a timeout of 1 hour per instance.

For all the different instances of the testing domains, we compile the set of human actions from the original GMDP $\mathcal{M}$ and transform it to a GMDP-HH $\mathcal{M}_H$. But, as discussed in previous chapters, the large number of human actions leads to a large branching factor in the search, which makes heuristic search less efficient. To overcome this issue, we select a subset $A'_H \subseteq A_H$ involving only the set of *relevant* fluents Göbelbecker et al. (2010); Helmert (2006), that is, the fluents that are relevant to lead the agent to the goal; these can be extracted from the domain description in PDDL Younes and Littman (2004). For the largest Triangle Tireworld instance, from a total of 92 fluents, we only used 46 relevant fluents to create the set of human help actions; for the largest Navigation instance, from a total of 309 fluents, only 154 were relevant fluents; for the largest Doors instance, from the total of 417 fluents, we only consider 146 relevant fluents.

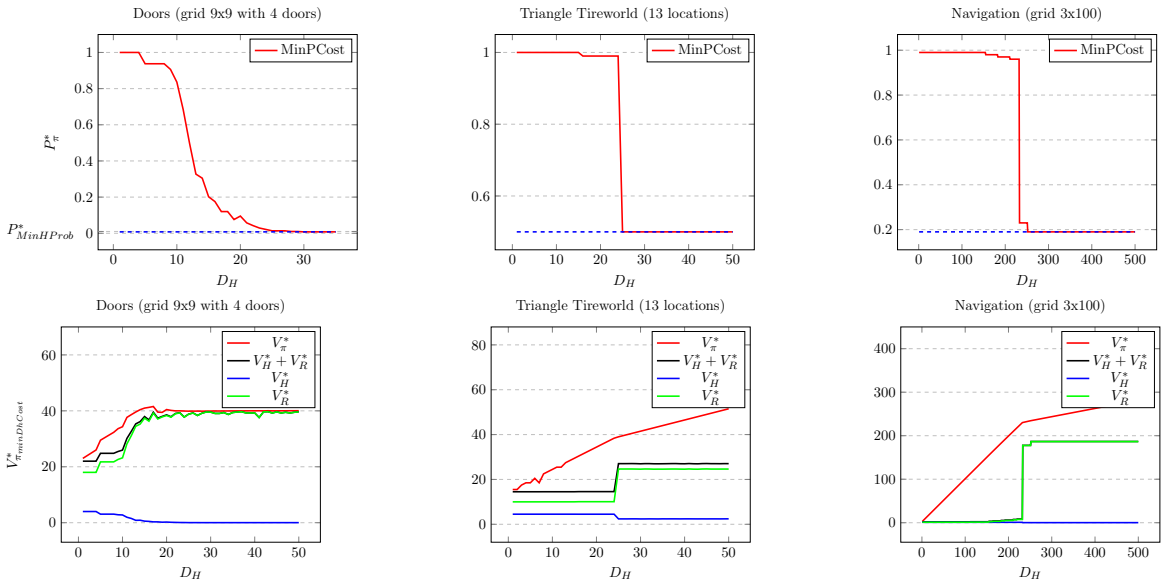


Figure 9.4: *Characteristics of the optimal policy for the three largest solved instances of the tested domains. Top:* $P^*(s_0) = P^{\pi_{MinPCost}}_{G_H}(s_0)$ *for increasing values of the penalty* D_H*;* $P^*(s_0) = P_{MinHProb}(s_0)$ *is the minimal probability (found for penalties* D_H *greater than 24, 25 and 250, respectively). Bottom:* $V_{MinPcost}(s_0)$*,* $V^*_R(s_0)$*,* $V^*_H(s_0)$ *and* $V^*_R(s_0) + V^*_H(s_0)$ *for increasing values of the penalty* D_H*.*

The human help actions allowed for the *Doors* domain consist in asking the human to give the key. More precisely, when the agent discovers that the location of the key is outside the room, and out of reach, it asks the human to put the key in some location near the agent. Making true the location of the key is a *static change*, because once the value of a key in a position x is set to false, for example when discovering that the key is outside the room makes false all the possible locations of a key, since there is no way to make these values true again, that is, there is no path in the *Domain Transition Graph* from the false value to the true value for this fluent, this is considered a static change. For the Triangle tireworld, the human help action is to give a spare wheel to the car. This static change is easy to see, because the spare tires can be consumed but they cannot be created, hence adding a spare tire in the location of the agent is a static change. However, in the *Navigation* domain, since there is only one variable encoding the location of the robot, and the *dead-end* consists of the robot disappearing, there are no static changes. A partial solution to this is considering the human can make true all of the relevant variables, that is, the human can put the robot in all the relevant variables.

Figure 9.3 shows the result of the three different criteria considered: *MinHProb*, *MinPCost* and *MinUCost* for the smallest instance of the domain Doors, consisting in a grid 5x5 with 2 doors. It is shown that the solution obtained with the *MinHProb* criteria, obtains the minimal probability independent of the cost of the human help actions.

Figure 9.4 shows the results of applying our solutions to an instance of each domain: Doors (grid 9x9 with 4 doors); Tireworld (triangle with 13 locations at each side); and Navigation (grid 3x100). For the Triangle Tireworld instance, from a total of 92 fluents, we only used 46 relevant fluents to create the set of human help actions; for the Navigation instance, from a total of 309 fluents, only 154 were relevant fluents; and for the Doors instance, from the total of 417 fluents, we only consider 146 relevant fluents.

Figure 9.4 (top) shows that, on one hand, the probability to use human actions from s_0 decreases when human action cost and penalty increase, until it reaches the minimum probability *MinHProb*, i.e., $P^{\pi^*}_{G_H}(s_0) = 0.0078$ for Doors instance, $P^{\pi^*}_{G_H}(s_0) = 0.5$ for Triangle Tireworld instance and $P^{\pi^*}_{G_H}(s_0) = 0.19$ for the Navigation instance. On the other hand, the optimal expected cumulative cost $V^{\pi^*}(s_0)$ increases (Figure 9.4 (bottom)), and converges to a maximum cost for Doors and Triangle Tireworld instances. This result was expected, since the probability to use human actions

Problem Instance	D_H	$P_{G_H}^{\pi_{MinPCost}}(s_0)$	$V^{\pi_{MinPCost}}(s_0)$	Time (sec)
Doors-7	30	0.03	31.6	3.4
Doors-9	30	0.01	39.4	5.4
Triangle Tireworld-4	50	0.50	49.6	0.6
Triangle Tireworld-5	50	0.50	57.6	3.0
Triangle Tireworld-6	70	0.50	74.0	65.4
Triangle Tireworld-7	90	0.50	90.5	757.0
Navigation 3x103	500	0.19	321.3	15.4
Navigation 4x103	500	0.27	381.9	19.5
Navigation 5x103	500	0.34	435.7	32.7

Table 9.3: *$P_{G_H}^{\pi_{MinPCost}}(s_0)$, $V^{\pi_{MinPCost}}(s_0)$, and time in seconds for the optimal policies, π^* computed with MinPCost criteria for a given penalty D_H.*

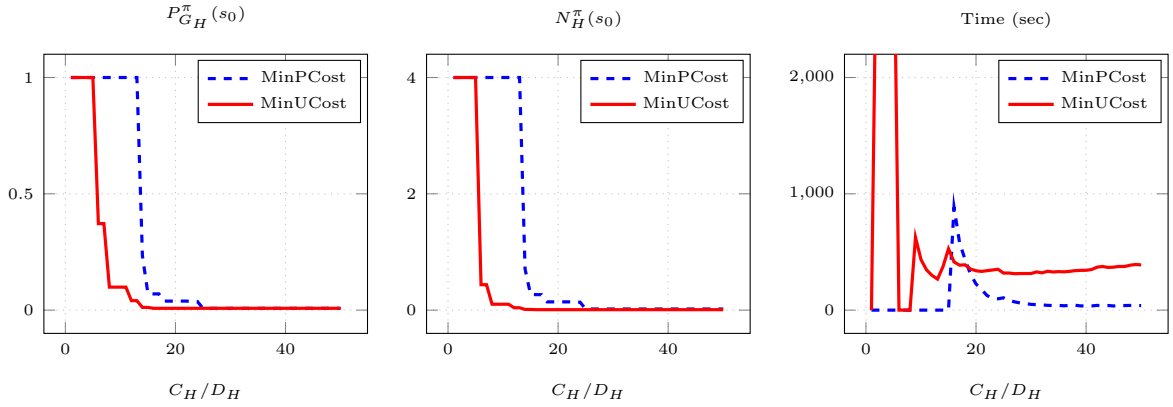


Figure 9.5: *Performance of different criteria as we increase the cost/penalty for the domain Doors (grid 9x9 with 4 doors).*

for the Tireworld domain is zero, and for the Doors domains is close to zero (0.0078). But in the Navigation domain, since $P_{G_H}^{\pi^*}(s_0) = 0.19$, the expected cost grows linear.

It is interesting to note that for the Doors domain this minimal probability is 0.00778 and the fixed cost to obtain this policy is 25, for the Triangle Tireworld the probability is 0.5 and the costs 25, but for the Navigation domain this probability is a bit higher 0.19 and the cost is around 250. Also notice how the costs converge for the Doors domain when the minimal probability is reached, because this minimal probability is near zero, increasing costs of the penalty will result in no appreciable increase of the expected cost of the optimal policy.

Table 9.3 shows the values of $P_{G_H}^{\pi_{MinPCost}}$, $V^{\pi_{MinPCost}}$, a large enough value of D_H that ensures convergence to the MinHProb policy and the time in seconds for finding the optimal policies under the *MinPCost* criteria, for 9 instances of the tested domains. In all instances, an estimate for the MinHProb policy was found with a sufficiently large value of D_H, and then compared with the analytically computed value. We see from the table, that most instances finished in a few seconds; the exceptions being the largest Triangle Tireworld instances, which are considerably large than the other instances.

9.3 Minimizing the Frequency of Human Actions

Figure 9.5 and Figure 9.6 show how the different criteria behave in two large planning instances: Doors-9 (grid 9x9 with 4 doors) and the Tireworld-4 (triangle with 9 locations at each side), in terms of $P_H^{\pi^*}(s_0)$, $N_H^{\pi^*}(s_0)$ and convergence time for varying values of human cost c_H and penalty D_H. Figure 9.5 (left) shows for the Doors-9 instance, that the probability to use human actions

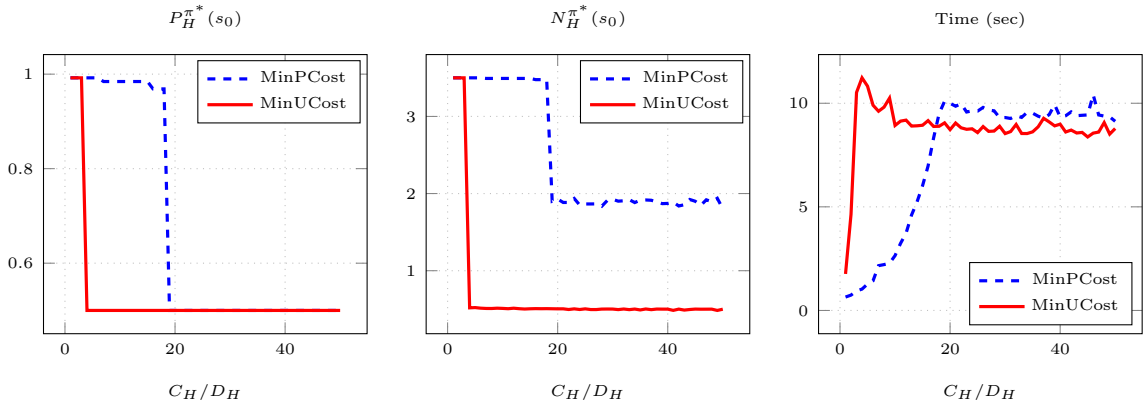


Figure 9.6: *Probability of using human help for the different criteria as a function of cost and penalty for the domain Triangle Tireworld with 9 locations.*

Problem Instance	MinUCost				MinPCost			
	$P_H^{\pi^*}$	$N_H^{\pi^*}$	V^{π^*}	Time(s)	$P_H^{\pi^*}$	$N_H^{\pi^*}$	V^{π^*}	Time(s)
Doors-7	0.03	0.03	32	240	0.03	0.08	31	5
Doors-9	0.01	0.01	36	805	0.01	0.02	36	34
Triangle Tireworld-5	0.50	0.50	32	5	0.50	2.30	27	4
Triangle Tireworld-6	0.50	0.50	36	42	0.50	2.00	27	37
Navigation 3x103	0.19	0.23	222	10	0.19	0.36	188	17
Navigation 4x103	0.27	0.32	228	18	0.27	0.35	208	22
Navigation 5x103	0.34	0.44	232	33	0.34	0.46	209	41

Table 9.4: *$P_H^{\pi^*}$, $N_H^{\pi^*}$, V^{π^*} and time in seconds for the optimal policies, π^* computed with MinUCost and MinPCost criteria.*

from s_0 decreases when c_H and D_H increase, until it reaches the minimum probability *MinPCost* ($P_H^{\pi^*}(s_0) = 0.0078$ for Doors and $P_H^{\pi^*}(s_0) = 0.5$ for the Triangle Tireworld-4). The same happen to Tireworld-4, Figure 9.6 (right). For both domains the expected number of actions also decrease reaching the minimum expected value of $N_H^{\pi^*}(s_0) = 0.0078$ using the MinUCost criterion, showing that with this criterion he human action is used only when necessary. However, the optimal policy obtained with *MinPCost* criterion uses more human actions $N_H^{\pi^*}(s_0) = 0.02$ (Figure 9.5 and Figure 9.6 middle). Notice that with the *MinUCost* criterion, $P_H^{\pi^*}(s_0)$ and $N_H^{\pi^*}$ converge faster when compared to *MinPCost* criterion. Notice also how, for both domains, the time grows exponentially (in some cases even surpassing our timeout) until convergence from which the time shows to be independent of the C_H or D_H.

Table 9.4 shows $P_H^{\pi^*}$, $N_H^{\pi^*}$, V^{π^*} (computing only the robot actions cost) and time in seconds for the optimal policies, π^* computed with *MinUCost* and *MinPCost* criteria, for 7 instances of the tested domains. The results show that for all instances, both criteria found the same values for $P_H^{\pi^*}$, however $N_H^{\pi^*}$ is smaller for all instances in the optimal policies computed using the with MinUCost criterion., while V^{π^*} is always larger, when compared with optimal policies with *MinPCost* criterion. This is because, in order to reach the goal, policies with a smaller number of human actions include a larger number of robot actions.

9.4 Differences between criteria

Finally, to illustrate the difference between both criteria, consider the *Dialog-Based Recommendation System* (DBRS) problem. This problems models an agent that has to help a human recommending restaurants or theaters by starting a dialog with the human, about her preferences.

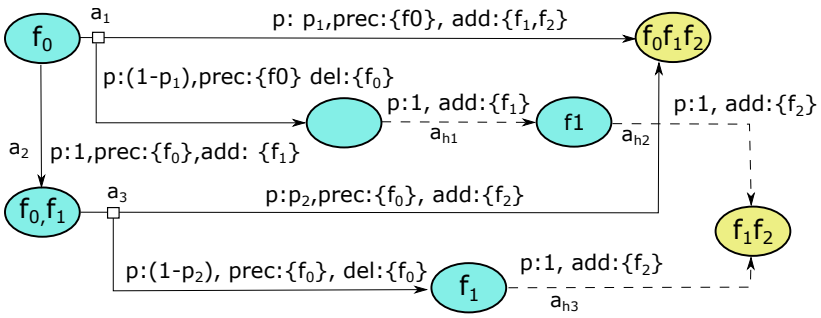


Figure 9.7: *Illustration of the Dialog-Based Recommendation System problem. Nodes, solid edges and dotted edges represent, respectively, states, agent actions and human help actions. Every actions is represented by the probability, the preconditions, and the add and del lists. The numbers close to edges denote the probability of the respective transition.*

The agent is able to make questions to the human, and to recognize the speech in the answer. But since there is an error in the interpretation, the agent can also ask the human to repeat the last answer if it is not well interpreted. In the case the answer is not understood at all, the agent can ask for human help, either by presenting a form the human must fill with the preferences for each question, or either by making the human select different options from a menu. We assume that for the human is easier to answer a question by speech than by filling a form or interacting with a touchscreen. Figure 9.7 shows a possible state space for a DBRS problem where the agent asks a composite questions, for example the address which is composed by a number and the name of the street. We denote by f_1 and f_2 the number and the name of the street respectively. There exists two possible ways to ask for these directions. In the first, the agent may ask directly to the user to tell the complete address (a_1). And there is a p_1 possibility that the agent recognize correctly the two fields (number and name). However there is also a possibility (1-p_1) that the agent fails to recognize both fields and may ask the human to write them down in an online form (a_{h1} and a_{h2}). In the second solution, the agent asks for the number and the name separately. In this case, the agent may recognize the number easily, but the agent may fail to recognize the name with a probability $1 - p_2$. But in this case, the form only asks one question (a_{h3}).

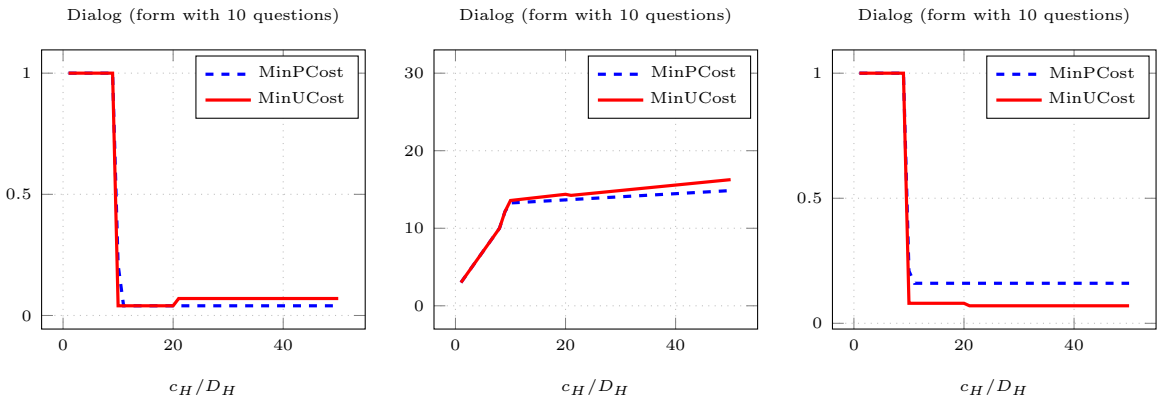


Figure 9.8: *Optimal expected probability of using Human Help $P_H^{\pi^*}$ (left), expected cumulative cost V^{π^*} (middle) and expected number of human actions $N_H^{\pi^*}$ (right) from s_0, increasing the cost c_H and the penalty D_H.*

In Figure 9.8 is showed the result of applying both criteria to this domain. The *MinPCost* converges to the minimal probability, but *MinUCost* do not converge to this probability. The expected cost is also different. Due to the *MinPCost* penalize the cost of the human actions only once, the cost of the other human actions is not accounted, as *MinUCost* does for every human action used. The last chart, shows the expected number of human actions, and it can be seen that even if *Min-*

Problem Instance	MinUCost			MinPCost		
	$V^{\pi^*}_{H_1}$	$V^{\pi^*}_{H_2}$	$V^{\pi^*}_{H_3}$	$V^{\pi^*}_{H_1}$	$V^{\pi^*}_{H_2}$	$V^{\pi^*}_{H_3}$
Doors-9	40.3	40.0	41.4	41.0	40.0	42.4
Triangle Tireworld-4	24.7	22.6	27.1	36.5	35.3	28.5
Navigation 3x100	40.3	40.0	41.4	40.9	40.0	42.4

Table 9.5: *Expected cumulative cost V^{π^*} for the optimal policies π^*, computing the cost of human actions using the human cost functions H_1, H_2 and H_3.*

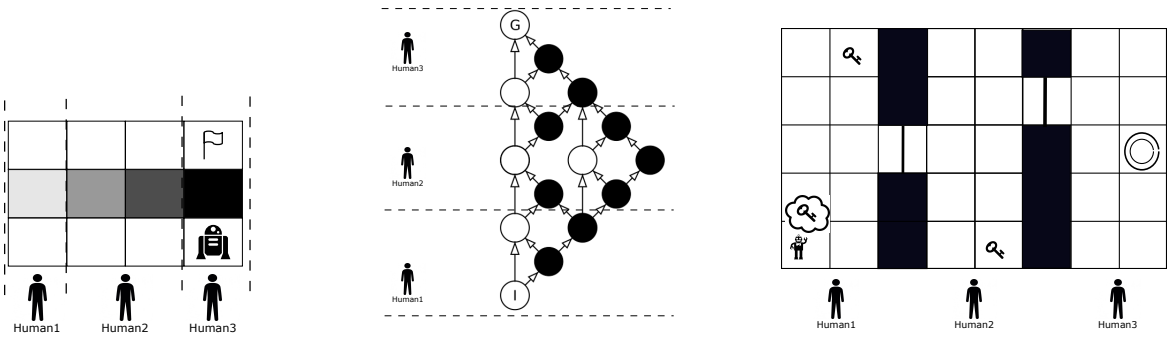


Figure 9.9: *Illustration of the three different domains considered, with three different humans $Human_1$, $Human_2$ and $Human_3$ in the scenario whose function costs are known. Each human has a reduced cost when it is required to help within its borders (dasehd lines), and its cost increases as it moves further from its region.*

UCost does not converge to the minimal probability, it does converge to the minimal number of expected human actions.

Finally, to test the robustness of both criterion where the human costs are known, we devised the following experiment, for one instance of each domain. In Table 9.4, $V_{\pi*}$ is computed without the cost of human actions, however we would like to know if the expected cost $V_{\pi*}$ changes for both criteria when the cost of the human actions is known. We considered three different humans in the scenario ($Human_1$, $Human_2$ and $Human_3$, Figure 9.9), each with its own set of actions and cost function, and tested the optimal policies computed with both criteria MinPCost and MinUCost. In Table 9.5 we compute V^{π^*} including the cost human actions, for three different humans ($Human_1$, $Human_2$ and $Human_3$), for one instance of each domain. The results show that computing the value of the policy computed with the *MinUCost* with a given human cost function (C_{H_i}), is always better then the one computed with *MinPCost* criterion.

9.5 Discussion

Algorithms that solve GMDPs are not able to prescribe actions when a dead-end is encountered. To overcome these situations, we defined a new class of problems called GMDP-HH, that generalizes SSPs and is able to use human help in the case the agent can not find a policy with probability 1 to reach the goal. We also proposed different optimization criteria and algorithms to solve GMDP-HHs that, unlike current solutions to deal with dead-ends, find a proper policy and minimize the probability of using human help. In this Chapter we have shown empirically how our planner, build upon these ideas, is able to obtain proper policies to solve GMDP with dead-ends by using human help.

The GMDP-HH with human actions that can modify the value of any fluent in any state is inspired by the work of finding good excuses to deterministic planning problems with no solution Göbelbecker et al. (2010). Thus, this work can also be used to: (i) giving to the human good excuses for the robot incompetence to accomplish its task, and (ii) giving some guidance to repair a planning domain description. Since the set of human actions in the optimal policy will satisfy the

optimization criteria proposed in this paper, they are certainly more complex excuses (repair) than the ones proposed by Göbelbecker et al. (2010).

Chapter 10

Related Work

According to the responsibility for finishing the task, human-robot collaboration can be divided in two areas: (1) symbiotic autonomy, where the robot is responsible for finishing the task, but it can ask for help; and (2) adjustable autonomy, where the human is equally responsible for finishing the task, but can delegate some tasks to the robot.

We have already explained symbiotic autonomy in Chapter 6, and in this chapter we will focus on approaches relying on MPD models as adjustable autonomy and Mixed-Initiative Planning. Adjustable autonomy solutions usually model the problem as a MDP, where a high level controller divides the tasks between the human and the robot. An example of such approach is the *Mixed-Initiative Markov Decision Processes* (MI-MDP) (Côté et al., 2012; Mouaddib et al., 2009, 2010).

In all previous approaches, the human actions are known a priori, but the novelty in our work is the generation of these actions from the GMDP problem description. In this chapter, we will look closely to different approaches to model the human intervention and the reasoning about human help, and how these solutions compare to our approach in terms of: what kind of help the human can offer, when this help is asked and who is responsible for the completion of the task.

10.1 Adjustable autonomy

An agent that is fully autonomous does not need human help to operate. On the other side, any robot teleoperated has no autonomy, and does not need to come up with a plan, since is the human who is controlling it. Between these two approaches stand the *Adjustable Autonomy* approach, that refers to agents changing dynamically its autonomy. These agents are able to act on their own, *transferring* decision making control to (generally) humans, in key situations.

One of the objectives of *Adjustable Autonomy* is to decrease human workload in human-robot interaction tasks. To effectively known where and how transfer autonomy, it is necessary to devise *transfer-of-control* strategies (Scerri et al., 2002), that is to dynamically change the decision maker. For example, an strategy AH is a strategy where the agent (A) initially takes control over the decision making, and when finds out that it is unable to make a decision, it passes the control to the human (H). In this section we discuss some of the approaches to this transfer of control, along with a discussion on the mode of adjustable autonomy.

10.1.1 Tranfer of Control Strategies

Scerri et al. (2002) introduces the first attempt to establish a theory of *transfer of control*. In their work agents can ask humans to take control. However since humans (usually) take more time to respond, delaying an answer is an important factor that must be considered, and it has a cost $\mathcal{W}$. The agent is also allowed to wait, by using the *deadline delaying* action $\mathcal{D}$, which reduces the cost $\mathcal{W}$ of waiting. Then, an example of strategy could be: $a_{bob}, a_{alice}, \mathcal{D}, a_{bob}$ in which the agent gives control to Bob, then to Alice, then inform that the decision has been delayed to finally give control to Bob indefinitely.

Formally, the task of *transfer of control* is defined by:

- an agent A;
- the decision d;
- S is the set of *transfer of control* strategies;
- n entities $e_1, e_2, ..., e_n$ who can make a decision, and they can be human or other agents;
- the expected rewards of the decisions make by the entities $EQ = \{EQ^d_{e_i}(t) : \mathcal{R} \to \mathcal{R}\}$, known by the agent;
- $P = \{P_\intercal(t) : \mathcal{R} \to \mathcal{R}\}$ is the continuous probability distribution that the in time t agent e will respond with a decision d of expected utility cost $EQ^d_e(t)$;
- cost of waiting a decision $\mathcal{W} : t \to \mathcal{R}$; and finally
- the *deadline delaying* action $\mathcal{D}$;

The wait cost function $\mathcal{W}$ is non-decreasing and there is some point in time where the costs stop accumulating.

The objective of an agent in the *transfer of control* task is to calculate the strategy $s \in S$ such that there is no other strategy s' with an expected reward higher than s, i.e. $\forall s' \in S, s' \neq s, EQ^d_s(t) \geq EQ^d_{s'}(t)$.

Some useful *general* strategies proven by the authors include:

- If the cost of waiting a decision $\mathcal{W}$ is increasing, while the agent is waiting for an answer, then a human response is better than autonomous decision. In the case that the agent has waited a lot, it is more interesting to wait for a human decision with a good reward than to let the agent decide and obtain a lower reward.
- $\mathcal{D}$ is useful if the expected value is greater than its cost, that is, if the agent knows that a human will respond and give a great reward, it is more interesting to wait for the reward.
- More $\mathcal{D}$ might not be necessary useful, because they may become redundant.

Expected rewards EQ depend on the cost of the waiting $\mathcal{W}$ and the value of a $\mathcal{D}$, and given the strategies showed above, the authors point out that it is still important to calculate the *transfer of control* strategies because waiting for the decision of a human U may overweight the expected reward, as opposed to the general strategy, hence making agent decision better. The timing of the transfer of control is also important because if the agent transfer control too early, the opportunity for a better decision may be lost and waiting for a better decision may incur in high waiting costs.

While our approach to minimize the human help can be seen also as a problem of finding an optimal transfer of control from the agent to the humans in the environment, there exist some difference between our approach:

- Our approach does not know neither the number of humans on the environment nor the actions these humans can perform;
- Our approach does not consider time.
- Even if our approach effectively switches control between the agent and some indefinite human on the environment, our approach gives the human a specific action to do, instead of only transferring control.

10.1.2 Autonomy Modes

When designing robots, a general rule for adjustable autonomy is that as autonomy increases, the breadth of the tasks that can be handled by a robot decreases (Crandall and Goodrich, 2001). Adjustable autonomy allows a human user to interact with a remote robot at different levels of autonomy: *fully-autonomous*, autonomous with *goal biases*, *waypoint methods*, *intelligent teleoperation* and dormant, that is totally controlled by human. Each of these levels, except dormant, is discussed below.

Fully autonomous is designed to let the robot interact with the environment of its own, based on its perceptions about the world. In this level, no human input is allowed to influence the robot, or is ignored.

In a system with *goal biased* autonomy, it is possible for a human to indicate a general direction, for example indicating a sub-goal the robot must reach in order to avoid some problematic region. The general indication can be positive, a region with positive rewards for visiting. Or negative, if the human wishes to specify a region that must be avoided. It can be seen that, even if the human specify general directions, leading the robot to a particular goal, the robot remains mostly autonomous. From a robot perspective, the only thing that the human changes is the reward function of some locations, leaving to the robot to calculate which are the best actions.

A *waypoint method* allows the human to indicate directions the robot must head to. This is a bit more intrusive that just making some regions more attractive (or repulsive) to the robot, as in *goal biased* autonomy. However, it requires a human more involved since the level of autonomy is reduced.

The teleoperation is the more mature approach (Sheridan, 1992), however, there may be obstacles to the communication when there are communication delays. A solution to this issue is to augment the autonomy, using supervisory control . But solutions that focus more on the interaction include Mixed-Initiative systems (Fong et al., 1999) and autonomy-based methods (Dorais et al., 1999).

Our approach is a bit different from these adjustable autonomy approaches. In our approach *full autonomy* is given to the robot. And only when it encounters a situation from where it cannot find a solution, it gives the control to the human. However, it does not give all control to the human at once. Instead it gives *waypoints* for the human be able to perform the minimal modification on the environment, and return control to the robot to continue acting until the goal is reached (or another intervention is needed).

10.1.3 Mixed-Initiative Planning

Mixed-Initiative control overcomes the limitations of autonomous robots introducing the human in the control loop. In a *mixed-initiative* environment, it is assumed that humans *could* perform all actions controlling the robot, but it will result in a time consuming ineffective solution. The key element in Mixed-Initiative systems is the dialogue between the human and the robot, and the fact that both parties share responsibility with each other for the success of the task (Crandall and Goodrich, 2001; Mouaddib et al., 2009, 2010)

In a *mixed-initiative* system, the delegation of the task can be initiated either by the robot (a *Response automation system*) or by the human (a *Task automation systems*) (Crandall and Goodrich, 2001). In the first case, the human chooses to delegate the task to relieve the work load. An example of this approach is a human setting waypoints for the robot. In the second case, the robot autonomously initiates the task to decrease the workload on the humans. An example of this approach is an interface that automatically delegates control to the robot when the human does not respond.

The automation will end if the assigned task is completed or the human operator intervenes. The limits of the automation provide two types of policies: *Management by exception* lets the robot take care of the automation. The robot, once given the control is completely responsible for the behavior of the system, and only returns the control if it encounters an exception or the human

intervenes. A robot that wanders while mapping a location is an example of this approach.

Management by consent lets the robot take care of the automation and gracefully return the control when the task is finished, or an exception is found. In this approach, the human does not know exactly the limits of the automation, and the robot must clearly indicates when it terminates. For example, a robot that must reach a location could give control back to humans if it has reached such location.

An approach to solve *mixed-initiative planning problems*, called *Mixed-Initiative Markov Decision Processes* (MI-MDP), consist in create an MDP with two sets of actions from the robot and the human Mouaddib et al. (2009, 2010). However, they differ in the type of actions and their costs. This cost should reflect the cost of time, and the human time is always more expensive that robot time. And even being capable of performing all actions of the robot, the human requires to *pay attention* to the task and reason about the situation before taking control. This level of attention to the activity depends on the complexity of the task, with more attention to the more complex tasks. To properly reflect this, the state space of a MI-MDP is augmented to indicate the attention of the human. Formally, a MI-MDP is a tuple $\langle S, A, R, T \rangle$, where:

- S is the state space. An MI-MDP state (s, i) includes the system state s and the level of human attention i;
- A is the set of actions, each action of the type $(a, *)$ or $(h, *)$ where a and h indicate an action of the robot or the human respectively, and $*$ indicates the variation on the level of attention, which can increase $(+)$, decrease $(-)$, go to level i or stay the same $(\emptyset)$;
- R is the *combined* reward function that assigns to each state (s, i) an immediate reward; and
- T is the *combined* transition model.

Solving the MI-MDP using an MDP solver, gives the optimal policy that shares control between the human and the robot. Côté et al. (2012) do not model the attention levels required of the humans, and the novelty of Côté et al. (2012) is how they detect online situations where the robot does is not able to find a solution. They focus on raising the *situational awareness* of the human, by giving extra information about the state where the robot failed to complete the task.

Solutions to *Mixed-initiative planning problems* are closer to our approach, however in our setting the human is not responsible for the success of the task, but the robot asks exactly what does it wants from the human. There are also some differences between our work and these approaches. The most important difference is that the actions the human is capable of performing are not know. Hence, the agent must plan and consents to give the human control only when no solution can be found (*Management by consent*). Since the robot provides the human with a full explanation for the failure, it is not required a high level of attention of the human.

10.2 Other approaches

Other works that use human help focus more on the bothering cost, that is the cost of interrupting the human and asking for help (Cohen et al., 2011; Levin et al., 2000). In general, to solve these problems they assign a different cost to the human actions and try to obtain the action (from the agent or the human) that minimizes the cost weighted by the probability of the human answering correctly (or at all). For example, in Levin et al. (2000) it is shown a model to manage an *Air Travel Information System* (ATIS) task, where an agent must minimize the length of a dialog with humans to obtain information. The agent in this problems is able to asks the human users open questions, or show them a form to fill with the correct date, and each one has a different cost (and probability of success). But in this case, they only assign different costs and solve it by means of a Value Iteration Algorithm.

Another example is the mission Deep Space One (DS1) (Bernard et al., 1998; Dorais et al., 1999). This unmanned spacecraft task has the objective of approaching an asteroid and a comet to

take pictures. The system that controls this spacecraft is called *Remote Agent Autonomous Control System*. This system controls the spacecraft 12 hours per day, generating plans for the spacecraft and recovering from failure. However, if the system can not correct the problem, the planner generates a new plan based on the diagnosed state of the spacecraft (Bernard et al., 1998). The DS1 remote system allows adjustable autonomy with three levels:

- Tele-operation from the ground controllers;
- Partially controlled from the ground, sending commands to be executed and recovering from failure; and
- Autonomously with some ground actions being executed and also executing a plan.

Conclusions and Future Work

Given the everyday growing scenario of using more and more technology to help humans to accomplish their tasks, an assumption that is often made is that softwares and robots must be completely autonomous and they should not fail under any circumstances. Although this must be true in security-critical systems, this is not (yet) a reality in many situations where it would be useful to have a robot (or software) to help the humans. Thus, defining a collaboration theory among humans and robots (or softwares) will bring up to our everyday life, even more interesting and useful technology applications.

Therefore, in this thesis we have proposed different frameworks to include human help in problems with uncertainty where there exists possibility for failure, either in contingent planning problems with partial observations and non-deterministic actions, or in planning problems with full observability and probabilistic effects. We have formalized this new problems and proposed solutions for them, creating agents that are able to pro-actively reason about using human observations and actions.

Thus, the main contributions of this thesis are:

- Formalization of contingent planning including human observations and actions automatically extracted from the problem description when the humans on the environment can not be modeled;
- A characterization of dead-end belief states for contingent planning;
- Use of the well-known planning structure, *Domain Transition Graph* and the *Causal Graph*, of a determinization of the planning problem (contingent or probabilistic), to produce a list of fluents that can be used to create a minimal set of human actions that guarantees the existence of strong (cyclic) solutions or proper solutions;
- Development of a contingent planning system with human help, called HH-CP planner that translates the contingent planning problem to a FOND problem using an epistemic translation to be solved by an efficient *off-the-shelf* FOND planner;
- Development of a contingent planning system with human help, called COMP2BT+HH, that extracts the smallest set of human actions that although incomplete, is more efficient than HH-CP;
- An empirical analysis of the HH-CP and COMP2BT+HH systems over a set of benchmark domains.
- Formalization of probabilistic planning problems including human actions, automatically extracted from the problem description.
- Formalization of probabilistic planning problems given a set of human actions.
- Definition of three different optimization criteria for probabilistic planning with human help to minimize the use of human help and an empirical analysis of their performance.

Model Extensions

The proposed model can be easily extended to account not only for humans in the environment, but also any external controller entity. For example, in a critical search and rescue mission, an aerial picture made by a drone is the only external help needed for the success of the mission. Thus, our model can be also used to create heterogeneous robotic teams, where each robot acts autonomously, but when needed it can asks help from the other robots.

Future Work

As a future work, it would be interesting for the contingent planning problems, not only to compute the policies that maximize the probability to reach the goal, but also to obtain the causes for these problematic situations, that is, the excuses for the robot's failure (Göbelbecker et al., 2010), that could be used: (1) to help the domain designer to modify the description of the problem; (2) as an explanation for the agent failure; and (3) to create actions specifically for human help to overcome these situations. If such situations where explicitly told in the description of the problem, for example given in the form of formulas that hold in any state that is a *dead-end*, or the competences of the humans were described as human actions applicable where the agent is no longer able to act or reach the goal, there exist already approaches that could obtain the best solutions.

On the probabilistic planning setting, we would like to obtain a Bellman Equation for the criterion of minimizing the expected number of human actions. This criteria should be a bit harder to find since it must combine in the same equation probabilities and the expected number of human actions.

Finally, we are also interested in extending our work to the POPP setting, that is, to consider probabilistic observations as in partially observable markov decission processes (POMDP). Such setting is considered to be one of the hardest because the belief states are no longer sets of possible states, but a continuous distribution of probability over all possible spaces.

Appendix A
POND Solutions

Conformant and contingent planning problems can be seen as a search problem in the space of beliefs instead of the space of states. But the space of beliefs is exponential in the number of states. While classical planning can be NP-Complete under certain restrictions (Bylander, 1994), the verification of existence of a plan is harder in the conformant setting, because the plan must be verified for every possible initial state.

Contingent problems are inherently hard to solve, and sometimes it is not desirable to compute the whole policy, but compute a single sequence of actions, and replan if any action of this sequence turns out to be not applicable. This approach is called *online* contingent planning. Online planners are focused on computing fast solutions, considering only one possible outcome for every observation and replanning when necessary. On the other hand offline planners compute the full policy considering all the outcomes of the observations. This allow these planners to compute better solutions, that might avoid dead-ends at the expense of the efficiency.

There exist different planning systems with different approaches to solve POND problems, for example Conformant-FF (Hoffmann and Brafman, 2006) and Contingent-FF (Hoffmann and Brafman, 2005) by performing an AND/OR search in the space of beliefs. However, these planners do not store the whole set of possible states, and only keep in memory all actions that have been applied and the initial belief state instead. When they need to test if a precondition holds before selecting an action, they use regression techniques to see if the formula of the precondition holds. Even if this planner was able to solve contingent planning problems in an offline fashion, the new approaches based on translations have rendered this planning obsolete.

Other examples of contingent planners include SDR (*Sample, Determinize, Replan*) (Shani and Brafman, 2011) and MPRS (*Multi Path Sampling Replanner*) (Brafman and Shani, 2012). The main idea behind the SDR planner is to randomly select a possible initial state $s \in b$ from the belief state, and consider it to be the real state. Thus, the problem becomes completely observable and a classical planner can be used to find a plan. This plan is executed as long as the actions a_i can be applied in the belief states b_i. If any action of the plan is no longer applicable, then an observation is performed and the rest of the plan is discarded. A new state is then selected from the updated belief state b. This process is repeated until a goal belief state is reached.

While SDR planner is able to tackle larger problems than offline planners, it is highly sensitive to *dead-ends*, because an agent can sample an state and follow a sequence of actions that result in the agent being in a state from where it is unable to reach the goal. To overcome this limitation, the MPRS planner is able to create a contingent plan in a linear form, that is, an ordered sequence of actions, that encode a contingent plan. To do so, it compiles the original contingent planning problem into a classical planning by including the preconditions of the actions into conditional effects, and causing that actions that are not applicable in the belief state b to not modify it. The agent then simply keeps applying the actions of the linear plan and it will reach the goal eventually. Even if the quality of these plans is worse, because it may create longer plans, these plans are safer in domains with *dead-ends*.

Those planners are confronted with two main challenges: the first is to keep track of what the agent knows to be true (or false), a task called *Belief Tracking*, and the second is to select which

is the best action to apply at each belief state. Both tasks can be intractable in the worst case (Bonet and Geffner, 2013). In this section we describe an approach, considered the state-of-the-art in conformant and contingent planning that solves POND problems in two phases: (1) it translates the original POND problem into an epistemic equivalent FOND problem; and (2) solve the FOND problem as an heuristic AND/OR search.

One of the first approaches to deal with the challenge of belief tracking divided the propositions of a planning problem into different sets, representing what is known by the agent with certainty, and what will be known after applying an observation (Petrick and Bacchus, 2002). This work was the basis for the new approach, called in this thesis k-translation, consisting in translating the planning problem with incomplete information into a planning problem with full information in the space of belief states, to be solved by a state-of-the-art planner for FOND problems, i.e., fully observable non-deterministic planning problem. Such translation can be viewed as an *epistemic* version of the original problem.

Next, we start by giving a brief introduction to Epistemic Logic, then we explain the different types of translations, pointing out their relations with some axioms of the Epistemic Logic. And we will finish by a discussion on the state-of-the-art planners that use this translation technique.

Appendix B
A brief introduction to Epistemic Logic

Epistemic logic allows an agent to reason about its knowledge of the world. (Fagin et al., 2003; Van Ditmarsch et al., 2007). Its basic version extends the propositional logic with an extra operator. Consider a propositional logic with a non-empty set of propositions $\{p, p', q, q'...\}$ describing basic facts of the world, e.g. p can represent "It is sunny in Brazil". Consider now the existence of a set of agents $\{a, b, ...\}$. To express facts of the type "Agent *a knows* it is sunny in Brazil", the language of propositional logic must be augmented with an unary epistemic operator K_a such that $K_a\phi$ is read as "agent a knowns ϕ", and it means that in all worlds known by the agent, the formula ϕ holds (Hintikka, 1962). In this section, for simplicity, and without loss of generality, we will consider worlds with a single agent. The basic assumption is that any knowledge involves dividing the set of possible worlds in two: Those worlds compatible with what is known and those that are incompatible with it.

Language of Epistemic Logic

The set of formulas that define the language of epistemic logic is given by Definition 32 :

Definition 32 (Basic epistemic language)**.** *Let P be a set of atomic propositions. The language for epistemic logic $\mathcal{L}_\mathcal{K}$ is generated by the following BNF:*

$$\phi ::= \top | p | \neg\phi | (\phi \wedge \phi) | K\phi,$$

where $p \in P$ is an atomic proposition, K is the Knowledge Operator, and $K\phi$ means "The agent knows that ϕ is true". ◇

A number of standard abbreviations are used throughout this section, such as: $\phi \vee \psi = \neg(\neg\phi \wedge \neg\psi)$; $\top$ stands for an abbreviation for $p \vee \neg p$ for an arbitrary $p \in P$; $\bot$ stands for $\neg\top$ and $(\phi \rightarrow \psi) = (\neg\phi \vee \psi)$.

The *semantics* of the epistemic logic, i.e. a formal model to determine if a given formula in the language is true or false, can be defined in terms of possible worlds (or states) formalized in terms of Kripke structures (Kripke, 1963). A Kripke structure M over P is a tuple $(S, \rho, \mathcal{K})$, where S is a set of possible worlds, ρ is an interpretation that associates each world $s \in S$ and each proposition $p \in P$ a truth value ($\rho : S \times P \rightarrow \{true, false\}$), and $\mathcal{K}$ is a binary relation on S, that is, a set of pairs of elements of S. Given two states of the world $s_1, s_2 \in S$, $(s_1, s_2) \in \mathcal{K}$ indicates that the agent considers s_2 to be possible given the information in world s_1, or in other words, that both states are indistinguishable. This relation is: (a) Reflexive, which means that $\forall s \in S$, $(s, s) \in \mathcal{K}$; (b) Symmetric, which means that $\forall s_1, s_2 \in S$, $(s_1, s_2) \in \mathcal{K}$ iff $(s_2, s_1) \in \mathcal{K}$ and (c) Transitive, which means that $\forall s_1, s_2, s_3 \in S$, if $(s_1, s_2) \in \mathcal{K}$ and $(s_2, s_3) \in \mathcal{K}$, then $(s_1, s_3) \in \mathcal{K}$.

A proposition, $p \in P$, is said to be true in a state $s \in S$ in M (written $M, w \models p$) iff $\rho(s, p) = true$. The semantics for the Boolean connectives follow the usual recursive recipe: $M, s \models \phi_1 \wedge \phi_2$ iff $M, s \models \phi_1$ and $M, s \models \phi_2$. And $M, s \models \neg\phi$ iff $M, s \not\models \phi$.

Formulas of the form $K\phi$ indicate that the agent knows ϕ to be true in a state s of structure M, if ϕ is true in all states that are deemed possible. Formally:

$$M, s_1 \models K\phi \text{ iff } M, s_2 \models \phi \text{ for all } s_2 \text{ such that } (s_1, s_2) \in \mathcal{K} \tag{10.1}$$

To illustrate these definitions, consider a simple example, the *SV*-scenario (an adaptation of the GLO-scenario from Van Ditmarsch et al. (2007)). Let us consider the following situation, where there is an agent a living in São Paulo, and it wants to reason about the weather conditions in both São Paulo and Valencia: in São Paulo is either sunny (p) or not ($\neg p$). The same applies for Valencia: sunny (v) or not ($\neg v$). In this setting, four different states of the world exist: both cities are sunny $\langle p, v\rangle$, sunny in São Paulo but not in Valencia $\langle p, \neg v\rangle$, sunny in Valencia but not in São Paulo $\langle \neg p, v\rangle$ and not sunny in neither of those cities $\langle \neg p, \neg v\rangle$. Since the agent a is based in São Paulo, it can be assumed that he knows the weather in São Paulo but not in Valencia, so the agent is unable to distinguish between states $\langle p, v\rangle$ and $\langle p, \neg v\rangle$, and likewise between $\langle \neg p, v\rangle$ and $\langle \neg p, \neg v\rangle$.

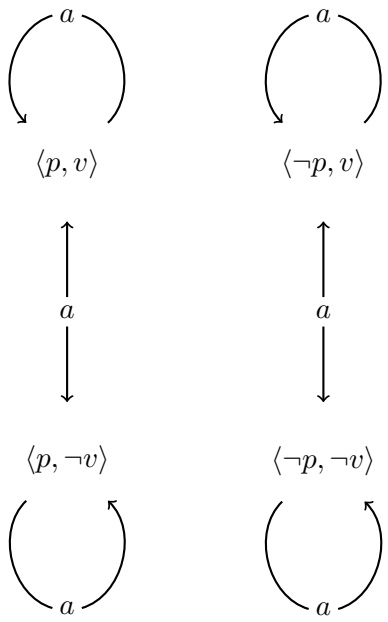


Figure 10.1: *Kripke Model for the SV-scenario. Nodes represent states, labeled with the value of the propositions, and there is an edge between two states, labeled with the agent's name if these states are indistinguishable for the agent, according to the relation $\mathcal{K}$.*

Figure 10.1 shows the Kripke model for the SV-scenario. The nodes represent the possible states, and node's labels describe which propositions are true and false in the state. Indistinguishable states are connected with edges labeled with the agent's name (a is the only agent considered in this example). An edge between two states $\langle p, v\rangle$ and $\langle p, \neg v\rangle$, can be read as "agent a cannot distinguish state $\langle p, v\rangle$ from state $\langle p, \neg v\rangle$", or more specifically "states $\langle p, v\rangle$ and $\langle p, \neg v\rangle$ can be the real state of the world as far as the agent knows".

To see if a formula $K\phi$ is true in a state s of structure M, formula 10.1 indicates that it must be true in all states that are deemed possible, i.e. indistinguishable from the state s. If the state of the world is $\langle p, v\rangle$, then agent a *knows* with certainty p (it is sunny in São Paulo), expressed as Kp, but the agent a does not know if it is sunny in Valencia v, because it cannot rule out the possibility of not being sunny $\neg v$. So, given the Kripke model of Figure 10.1 and the state $\langle s, v\rangle$, the following formula holds in that state: $Kp \wedge \neg Kv$.

Axioms of Epistemic Logic

An axiomatization is a syntactic way to characterize a logic. *Axioms* are a core set of formulas from which all other formulas in the logic are derivable. The axioms of epistemic logic are composed by 2 sets of axioms. The first set of axioms **K** forms a basic modal system. It is composed by the

Modus Ponens (**MP**) and two other axioms: the *Distribution Axiom* (**Prop**) and the *Necessitation Axiom* (**N**):

- The Distribution Axiom (**Prop**) states that each agent knows the logical consequences of his knowledge, i.e. if an agent knows ϕ and also knows that $\phi \rightarrow \psi$, then ψ must be also known in any state it considers to be possible $K\phi \wedge K(\phi \rightarrow \psi) \rightarrow K\psi$.
- The Necessitation Axiom (**N**) states that valid formulas in a state are known by the agents $\phi \rightarrow K\phi$. If a formula ϕ is valid in all states, then it must be true at all states that the agent considers possible. These are the formulas that are *necessarily* true, as opposed to the formulas that just happen to be true at a given state.

And the second set of axioms, comprise the Truth Axiom (**T**) and the agent's introspection regarding their own knowledge:

- The Truth Axiom (**T**) expresses that the knowledge of the agent is in fact true $K\phi \rightarrow \phi$. If the agent knows a fact ϕ, then this fact must be true: $K\phi$. This is the main difference between knowledge and belief. The agent may believe something and it can be a false belief, but the agent cannot know something that is false.
- The *Positive Introspection Axiom*, known as **4**, it states that the agent *knows that it knows what it knows*: if the agent knows ϕ to be true $K\phi$, it also knows that it knows that it knows ϕ, $KK\phi$.
- And the *Negative Introspection Axiom*, known as **5**, states that the agent *knows that it does know what it does not know*: if the agent does not know ϕ, $\neg K\phi$, then it also knows that it does not know it $K\neg K\phi$.

The axioms **K** with the axioms **T**+**4**+**5** are known as the **S5 Properties** (Van Ditmarsch et al., 2007). In the following section we introduce a technique used to solve POND problems that satisfy axioms **K** and **T**.

Appendix C
Example of an Incomplete K-Translation

To illustrate the soundness of the translation and why it is necessary to have the cancellation rules, consider the following translated classical problem $K(P) = \langle F, I, A, G\rangle$ (Palacios and Geffner, 2006) where:

- $F = \{Kp, Kq, Kr, Ks, Kt, Kg\}$;
- $I = \{Kr, Ks, \neg Kp, \neg Kq, \neg K\neg p, \neg K\neg q\}$;
- $G = \{Kt, Kg\}$; and
- $A = \{a, b\}$, with the conditional effects:
  - $e_1(a)$ $(support) : \langle\{Kp\}, \{K\neg r\}, \{\emptyset\}\rangle$,
  - $e_2(a)$ $(support) : \langle\{Ks\}, \{Kt\}, \{\emptyset\}\rangle$, and
  - $e_1(b)$ $(support) : \langle\{Kr\}, \{Kg\}, \{\emptyset\}\rangle$.

It can be seen that for this translated problem, both plans $\langle b, a\rangle$ e $\langle a, b\rangle$ are valid conformant plans. But in the real (physical) world, the second plan would never reach the goal, because f the uncertainty in the initial state, the value of p is not known in the initial state, that is, $\neg Kp$ and $\neg K\neg p$ are true in the initial state. Hence, it is not possible to know if all the effects of action a are applicable and what value will hold r or $\neg r$. If we add the cancellation rules to the problem, the conditional effects of actions a and b with conditional effects:

- $e_1(a)$ $(support) : \langle\{Kp\}, \{K\neg r\}, \{\emptyset\}\rangle$;
- $e_1(a)$ $(cancellation) : \langle\{\neg K\neg p\}, \{\emptyset\}, \{K\neg r\}\rangle$;
- $e_2(a)$ $(support) : \langle\{Ks\}, \{Kt\}, \{\emptyset\}\rangle$;
- $e_2(a)$ $(cancellation) : \langle\{\neg K\neg s\}, \{\emptyset\}, \{\neg K\neg t\}\rangle$;
- $e_1(b)$ $(support) : \langle\{Kr\}, \{Kg\}, \{\emptyset\}\rangle$; and
- $e_1(b)$ $(cancellation) : \langle\{\neg K\neg r\}, \{\emptyset\}, \{\neg K\neg g\}\rangle$.

Now, the execution of the sequence $\langle a, b\rangle$ will not lead the agent to the goal, because a will assign to the Kr fluent a value of false.

Appendix D
A complete K-Translation

$K_{T,M}$-Translation for Conformant Planning

The K_0-Translation is sound but not complete. To achieve completeness, the translation $K(\mathcal{M})$ can be extended to the translation $K_{T,M}(\mathcal{M})$ (Palacios and Geffner, 2007b) that introduces the concept of *tags* and *merges*.

A conformant planning problem $\mathcal{M}^{\text{CONT}}$ can be translated to a classical planning problem $K_{T,M}(\mathcal{M})$, where T and M are two parameters indicated before performing the translation: T is a set of *tags* and M is a set of *merges*. With a suitable set of *tags* and *merges*, the translation $K_{T,M}(P)$ can achieve completeness.

A *tag* $t \in T$ is a set of literals L from $\mathcal{M}^{\text{CONF}}$ whose value is unknown in the initial state I. The literals KL/t in the translation $K_{T,M}(\text{M})$ mean that the literal L is known to be true, if and only if t is true in the initial state.

K_{S0} is a particular case of translation that is sound and complete, and is obtained as follows:

- T is the set of all literals whose value is not know in the initial Belief state state I.
- M is the set of *merges* (T', L), being T' equal to T but without the empty *tag*. And being L the literals in P that are goals or preconditions of actions.

In the translation K_{S0} the initial uncertainty is substituted with different known initial states.

- K_{S0} **Completeness**: If π is a plan that solves the classic planning problem K_{S0}, then the sequence of actions π', that is equal to π but without the *merge* actions, is a plan that solves the conformant planning problem $\mathcal{M}^{\text{CONF}}$, and, if π is a conformant plan for $\mathcal{M}^{\text{CONF}}$, then exists a sequence of actions π' that extends π with *merge* actions and is a plan for K_{S0}.

The main problem of introducing literals KL/t, is the size of the problem that grows exponential in the worst case, resulting in all the propositions of the problems KL for every possible *tag* t. An example of this can be seen in the *Localize* problem showed in Figure 3.10. In this conformant problem, the initial position of the agent is unknown and it could be located in every position of the map, hence there is a *tag* t for every for every valid position on the map. This results in the literals $loc(x, y)$ and $\neg loc(x, y)$ multiplied by all the possible locations: $Kloc(x, y)/t_i$ $K\neg loc(x, y)/t_i$ where t ranges over all locations $loc(x, y)$. A complete translation would be exponential in the number of possible initial states.

$K_{T,M}$-Translation for Contingent Planning

So far, we have seen the translation approach to solve conformant planning problems. In this section we explain the extension of this translation to be used in contingent planning problems. Let us remember that a contingent planning problem is a planning problem with uncertainty in the initial state, non-deterministic actions and observations, that can be seen as a non-deterministic search problem in the space of beliefs (Hoffmann and Brafman, 2005).

As showed before, recent approaches to solve conformant planning problems (Palacios and Geffner, 2007b, 2009) use these translations to compile the original problem into a classical problem and then use a state-of-the-art Classical planner as FF-Planner, to solve the problem.

However this approach presents two limitations: non-deterministic actions and types of solutions. In the first place, this approach is limited to contingent planning problems with uncertainty in the initial state and deterministic actions only, limiting the non-deterministic outcomes only for the sensing actions. And in the second place, is not possible to solve the translated contingent planning problem using a state-of-the-art Classical planner, because the structure of the solution is different in both cases. In fact, plans that solve classical planning problems are sequences of actions, whereas in contingent planning it is a contingent tree, or a policy (see Section 3.2.2), mapping states into actions. If an action prescribed by the policy $\pi(b_n)$ is an observation, then the node labeled as b_n that represents the belief state two children, one for each outcome of the observation. A policy tree solves a problem $\mathcal{M}^{\text{CONT}}$ if the branches of the tree represent sequences of actions that are applicable and end in a belief state b_G that satisfies G.

The basic translation $X(\mathcal{M}) = X_{T,M}(\mathcal{M})$ of a contingent planning problem is defined as the translation $K_{T,M}(\mathcal{M})$ of the conformant part of $\mathcal{M}^{\text{CONT}}$, i.e. without considering the observations, extended with two additional components: the conditional observations and a set of deductive rules to complement the conformant *merges* and to infer which *tags* are meant to be false when they contradict the observations. Observations are translated as non-deterministic actions, where the value of the observed L is obtained, simulating the effect of observing the real value of L:

$$obs(L) : \neg KL \wedge \neg K\neg L \rightarrow KL|K\neg L. \tag{10.2}$$

The set of additional deductive rules represent the contingent *merge* that now is generalized with respect to the conformant version. To know the value of L, it suffices to known KL/t for all the non-refuted *tags*:

$$\bigwedge_{t\in m, m\in M_L} (KL/t \vee K\neg t) \rightarrow KL$$

The refuted *tags* are those that predict a literal L that has been observed false:

$$KL/t \wedge K\neg L \rightarrow K\neg t$$

Example

As an example let us consider the *Contingent Medical problem* $\mathcal{M}^{\text{CONT}}$ (Albore et al., 2009):

- $I = \{\neg s\}$, and the goal $G = \{h\}$.
- Actions a and b, with preconditions $a : d$ and $b : \neg d$, both with effect h. And action c with conditional effect $e : \langle\{d\}, \{s\}, \{\emptyset\}$.
- Sensing action $obs(s)$, without precondition.

A valid contingent plan for this problem can be $\Pi = \{c, obs(s),$ if true a else $b\}$. The plan above can be seen as a tree with two branches of sequences of actions and observations $\pi_1 = \{c, o^+(s), a\}$ and $\pi_1 = \{c, o^-(s), b\}$, where $o^+(s)$ and $o^-(s)$ are the observations x and $\neg x$ respectively. Translating the problem $\mathcal{M}^{\text{CONT}}$ using translation $X_{T,M}(\mathcal{M})$ and using the set of *tags* $t_1 = \{d\}$ and $t_2 = \{\neg d\}$.

The plan will work as follows: action c in the first ramification will render $K\neg s/t_2$ true, while $o^+(s)$ render true the literal Ks. Refuting the *tag* it is obtained $K\neg t_2$, that with Kd/t_1 that is initially true, will render Kd from the contingent *merge*. And the second ramification will work in a similar way.

Bibliography

Alexandre Albore. *Translation-based approaches to automated planning with incomplete information and sensing*. PhD thesis, Universitat Pompeu Fabra, 2012. 35

Alexandre Albore and Héctor Geffner. Acting in partially observable environments when achievement of the goal cannot be guaranteed. In *Proceedings of ICAPS Workshop on Planning and Plan Execution for Real-World Systems*, 2009. 63

Alexandre Albore and Hector Geffner. Effective heuristics and belief tracking for planning with incomplete information. In *In Proceedings of the 21st Int. Conf. on Automated Planning and Scheduling*, pages 2–9, 2011. 39

Alexandre Albore, Héctor Palacios, and Héctor Geffner. A Translation-Based Approach to Contingent Planning. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI).*, pages 1623–1628, 2009. 19, 41, 53, 57, 58, 124

Ignasi Andrés and Leliane Nunes de Barros. Using the Causal Graph to Enhance Translations to Solve Contingent Planning Problems. In *Proceedings of the 5th Brazilian Conference on Intelligent Systems, BRACIS 2016, Recife, Brazil, October 9-12, 2016*, 2016. 5, 50, 51

Ignasi Andrés, Leliane Nunes de Barros, and Karina Valdivia. Asking Human Help in Contingent Planning. In *First International Workshop on Teams in Multiagent Systems (TEAMAS) 2017, São Paulo, Brazil, May 9, 2017*, pages 69—81, 2017. 5, 49, 58

Ignasi Andrés, Leliane Nunes de Barros, Denis Maua, and Dias Thiago. When a Robot Reaches Out for Human Help. In *Proceedings of the 16th Ibero-American Conference on Artificial Intelligence (IBERAMIA) 2018, Trujillo, Peru, 2018*, 2018a. 5

Ignasi Andrés, Leliane Nunes de Barros, and Karina Valdivia. Human Aware Contingent Planning. *Under revision*, 2018b. 5

Andrew Tarantola. Robot caregivers are saving the elderly from lives of loneliness. https://www.engadget.com/2017/08/29/robot-caregivers-are-saving-the-elderly-from-lives-of-loneliness/, 2017. [Online; accessed 19-July-2018]. 1

Nicholas Armstrong-Crews and Manuela Veloso. Oracular partially observable Markov decision processes: A very special case. In *Proceedings of the IEEE International Conference on Robotics and Automation.*, pages 2477–2482. IEEE, 2007. 5, 63, 64

Christer Bäckström and Bernhard Nebel. Complexity results for sas+ planning. *Computational Intelligence*, 11:625–656, 01 1995. 11

Andrew G Barto, Steven J Bradtke, and Satinder P Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1-2):81–138, 1995. 76

Richard Bellman and Robert Kalaba. On the role of dynamic programming in statistical communication theory. *IEEE Transactions Information Theory*, 3(3):197–203, 1957. 28

D. E. Bernard, G. A. Dorais, C. Fry, E. B. Gamble, B. Kanefsky, J. Kurien, W. Millar, N. Muscettola, P. P. Nayak, B. Pell, K. Rajan, N. Rouquette, B. Smith, and B. C. Williams. Design of the remote agent experiment for spacecraft autonomy. In *Proceedings of the IEEE Aerospace Conference*, volume 2, pages 259–281. IEEE, 1998. 110, 111

Dimitri P Bertsekas and John N Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16(3):580–595, 1991. 3, 19, 73, 75

Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial intelligence*, pages 1–20, 1997. 12

Blai Bonet. Labeled RTDP: Improving the convergence of real-time dynamic programming. In *In Proceedings of the 13th International Conference on Automated Planning and Scheduling. ICAPS-03*, 2003. 19, 75, 76, 94, 99

Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129, 2001. 12, 18

Blai Bonet and Héctor Geffner. Faster heuristic search algorithms for planning with uncertainty and full feedback. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 1233–1238, 2003. 3, 75, 77, 94

Blai Bonet and Héctor Geffner. An Algorithm Better Than AO*? In *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 3*, AAAI'05, pages 1343–1347. AAAI Press, 2005a. 28

Blai Bonet and Héctor Geffner. mGPT: A probabilistic planner based on heuristic search. *Journal of Artificial Intelligence Research*, 24(1):933–944, 2005b. 99

Blai Bonet and Héctor Geffner. Planning under partial observability by classical replanning: Theory and experiments. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 1936, 2011. 34, 41

Blai Bonet and Héctor Geffner. Causal belief decomposition for planning with sensing: completeness results and practical approximation. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 2275–2281. AAAI Press, 2013. 39, 116

Blai Bonet and Héctor Geffner. Belief tracking for planning with sensing: Width, complexity and approximations. *Journal of Artificial Intelligence Research*, pages 923–970, 2014a. 34, 37

Blai Bonet and Héctor Geffner. Flexible and Scalable Partially Observable Planning with Linear Translations. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI)*, pages 2235–2241. AAAI Press, 2014b. 39

Ronen I Brafman and Guy Shani. A Multi-Path Compilation Approach to Contingent Planning. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI)*, 2012. 54, 57, 63, 115

Tom Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1):165–204, 1994. 115

Chenghui Cai, Xuejun Liao, and Lawrence Carin. Learning to explore and exploit in POMDPs. In *Advances in Neural Information Processing Systems*, 2009. 5

Alberto Camacho, Christian J. Muise, and Sheila A. McIlraith. From FOND to robust probabilistic planning: Computing compact policies that bypass avoidable deadends. In *Proceedings of the 26th International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016.*, pages 65–69, 2016. 82

Alessandro Cimatti, Marco Pistore, Marco Roveri, and Paolo Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147, 2003. 18

Robin Cohen, Hyunggu Jung, Michael W Fleming, and Michael YK Cheng. A user modeling approach for reasoning about interaction sensitive to bother and its application to hospital decision scenarios. *User Modeling and User-Adapted Interaction*, 21(4-5):441–484, 2011. 110

Nicolas Côté, Arnaud Canu, Maroua Bouzid, and Abdel-Iillah Mouaddib. Humans-robots sliding collaboration control in complex environments with adjustable autonomy. In *Proceedings of the The 2012 IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technology-Volume 02*, pages 146–153, 2012. 2, 6, 63, 87, 107, 110

Jacob William Crandall and Michael A. Goodrich. Experiments in adjustable autonomy. In *2001 IEEE International Conference on Systems, Man and Cybernetics. e-Systems and e-Man for Cybernetics in Cyberspace*, volume 3, 2001. 2, 6, 109

Gregory Dorais, R Peter Bonasso, David Kortenkamp, Barney Pell, and Debra Schreckenghost. Adjustable autonomy for human-centered autonomous systems. In *Sixteenth International Joint Conference on Artificial Intelligence Workshop on Adjustable Autonomy Systems*, pages 16–35, 1999. 109, 110

Finale Doshi, Joelle Pineau, and Nicholas Roy. Reinforcement learning with limited reinforcement: Using Bayes risk for active learning in POMDPs. In *Proceedings of the 25th International Conference on Machine learning*, 2008. 5

Stefan Edelkamp and Malte Helmert. Exhibiting knowledge in planning problems to minimize state encoding length. In *Proceedings of the 5th European Conference on Planning: Recent Advances in AI Planning*, ECP '99. Springer-Verlag, 2000. 48, 50

Mica R. Endsley. Design and Evaluation for Situation Awareness Enhancement. *Proceedings of the Human Factors Society Annual Meeting*, 1988. 6

European Comission. Public attitudes towards robots. http://ec.europa.eu/commfrontoffice/publicopinion/archives/ebs/ebs_382_fact_dk_en.pdf, 2012. [Online; accessed 19-July-2018]. 1

Ronald Fagin, Joseph Halpern, Yoram Moses, and Moshe Vardi. *Reasoning About Knowledge*, volume 1. The MIT Press, 2003. 40, 117

Richard E Fikes and Nils J Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 1972. 9

Terrence W. Fong, Chuck Thorpe, and C. Baur. Collaborative control: A robot-centric model for vehicle teleoperation. In *Spring Symposium on Agents with Adjustable Autonomy*, 1999. 109

Ignasi Andrés Franch. Asking Human Help in Contingent Planning. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '17, pages 1832–1833, 2017. 5, 49, 58

Frank Tobe. Where Are the Elder Care Robots? https://spectrum.ieee.org/automaton/robotics/home-robots/where-are-the-eldercare-robots, 2012. [Online; accessed 19-July-2018]. 1

Héctor Geffner and Blai Bonet. A Concise Introduction to Models and Methods for Automated Planning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 2013. 3, 8, 10, 28

Georgia Institute of Technology. Robots in the home: Will older adults roll out the welcome mat? https://www.sciencedaily.com/releases/2012/10/121025161518.htm, 2012. [Online; accessed 19-July-2018]. 1

Malink Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practise*. Elsevier, 2004. xiii, 2, 7, 8, 16, 18

Moritz Göbelbecker, Thomas Keller, Patrick Eyerich, Michael Brenner, and Bernhard Nebel. Coming up with good excuses: what to do when no plan can be found. In *Proceedings of the Twentieth International Conference on International Conference on Automated Planning and Scheduling (ICAPS)*, pages 81–88. AAAI Press, 2010. 45, 49, 99, 104, 105, 114

Robert P Goldman and Mark S Boddy. Expressive Planning and Explicit Knowledge. In *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems (AIPS)*, pages 110–117. AAAI Press, 1996. 18, 33

Eric A. Hansen and Shlomo Zilberstein. LAO: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1-2):35–62, 2001. 94

Malte Helmert. A Planning Heuristic Based on Causal Graph Analysis. In *Proceedings of the 14th International Conference on International Conference on Automated Planning and Scheduling*, volume 16 of *ICAPS'04*, pages 161–170. AAAI Press, 2004. 16

Malte Helmert. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26:191–246, 2006. 12, 14, 15, 16, 47, 49, 99

J Hintikka. *Knowledge and belief: an introduction to the logic of the two notions*. Contemporary philosophy. Cornell University Press, 1962. 117

Jörg Hoffmann and Ronen I Brafman. Contingent planning via heuristic forward search with implicit belief states. In *Proceedings of the 15th International Conference on International Conference on Automated Planning and Scheduling (ICAPS)*, pages 71–88. AAAI Press, 2005. 3, 19, 115, 123

Jörg Hoffmann and Ronen I Brafman. Conformant planning via heuristic forward search: A new approach. *Artificial Intelligence*, 170(6):507–541, 2006. 18, 115

Jörg Hoffmann and Bernhard Nebel. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research*, 2001. 12, 14, 41, 51

Robin Jaulmes, Joelle Pineau, and Doina Precup. Active learning in partially observable Markov Decision Processes. In *Proceedings of the European Conference on Machine Learning*, 2005. 5

Johou Systems Kougaku (JSK) Laboratory. Home Assistant Robot AR. http://www.jsk.t.u-tokyo.ac.jp/research/irt/ar.html, 2012. [Online; accessed 03-August-2018]. 1

Peter Jonsson and Christer Bäckström. State-variable planning under structural restrictions: Algorithms and complexity. *Artificial Intelligence*, 100, 1998. 11, 15

Abir-Beatrice Karami, Laurent Jeanpierre, and Abdel-Illah Mouaddib. Partially Observable Markov Decision Process for managing Robot Collaboration with Human. In *21st IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 518–521, 2009. 5, 63

Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49, 2002. 5

Andrey Kolobov, Mausam, and Daniel S. Weld. SixthSense: Fast and Reliable Recognition of Dead Ends in MDPs. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, 2010. 78

Andrey Kolobov, Mausam Mausam, Daniel S Weld, and Héctor Geffner. Heuristic search for generalized stochastic shortest path MDPs. In *Proceedings of the 21th International Conference on Automated Planning and Scheduling*, 2011. 82, 83, 90

Andrey Kolobov, Mausam Mausam, and Daniel Weld. A theory of goal-oriented MDPs with dead ends. In *Proceedings of the 28th Conf. on UAI*, pages 438–447, 2012. 3, 78, 79, 80, 81, 82, 87, 91

Saul A Kripke. Semantical Analysis of Modal Logic I. Normal Modal Propositional Calculi. *Mathematical Logic Quarterly*, 9(5-6):67–96, 1963. 117

Esther Levin, Roberto Pieraccini, and Wieland Eckert. A stochastic model of human-machine interaction for learning dialog strategies. *IEEE Transactions on speech and audio processing*, 8 (1):11–23, 2000. 110

Abdel-Illah Mouaddib, Shlomo Zilberstein, and Aurélie Beynier. Mixed-initiative decision-theoretic planning for cooperative control. In *Journées Francophones Planification Décision Apprentissage*, 2009. 2, 6, 63, 107, 109, 110

Abdel-Illah Mouaddib, Shlomo Zilberstein, Aurélie Beynier, and Laurent Jeanpierre. A decision-theoretic approach to cooperative control and adjustable autonomy. In *Proceedings of the 2010 conference on ECAI 2010: 19th European Conference on Artificial Intelligence*, pages 971–972. IOS Press, 2010. 2, 6, 63, 87, 107, 109, 110

Christian Muise, Sheila A McIlraith, and J Christopher Beck. Improved non-deterministic planning by exploiting state relevance. In *Proceedings of the 22th International Conference on International Conference on Automated Planning and Scheduling (ICAPS)*, pages 172–180. AAAI Press, 2012. 18, 31, 32, 42, 49

Christian Muise, Vaishak Belle, and Sheila A McIlraith. Computing contingent plans via fully observable non-deterministic planning. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, pages 2322–2329. AAAI Press, 2014. 41, 42, 57

Allen Newell, John C Shaw, and Herbert A Simon. Report on a general problem-solving program. In *Proceedings of the International Conference on Information Processing*, pages 256–264, 1959. 7

Héctor Palacios. *Translation-based approaches to conformant planning.* PhD thesis, Universitat Pompeu Fabra, 2009. 34, 40

Héctor Palacios and Héctor Geffner. Compiling uncertainty away: Solving conformant planning problems using a classical planner (sometimes). In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1*. AAAI Press, 2006. 39, 40, 45, 121

Héctor Palacios and Héctor Geffner. From Conformant into Classical Planning: Efficient Translations that May Be Complete Too. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, September 22-26, 2007*, pages 264–271, 2007a. 45

Héctor Palacios and Héctor Geffner. From Conformant into Classical Planning: Efficient Translations that May Be Complete Too. In *ICAPS*, pages 264–271, 2007b. 18, 40, 123, 124

Héctor Palacios and Héctor Geffner. Compiling uncertainty away in conformant planning problems with bounded width. *Journal of Artificial Intelligence Research*, 35, 2009. 41, 124

Ronald P A Petrick and Fahiem Bacchus. A Knowledge-Based Approach to Planning with Incomplete Information and Sensing. In *AIPS*, pages 212–222, 2002. 116

Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming.* John Wiley & Sons, 1994. 19, 71

Diederik M Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48:67–113, 2013. 81

Stephanie Rosenthal, Joydeep Biswas, and Manuela Veloso. An effective personal mobile robot agent through symbiotic human-robot interaction. In *Proceedings of the AAMAS*, pages 915–922, 2010. 6

Stephanie Rosenthal, Manuela Veloso, and Anind K Dey. Learning accuracy and availability of humans who help mobile robots. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence*, pages 1501–1506. AAAI Press, 2011. 5, 63, 64, 65

Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010. 35

Scott Sanner. Relational Dynamic Influence Diagram Language (RDDL): Language Description. http://users.cecs.anu.edu.au/s̃sanner/IPPC_2011/RDDL.pdf, 2010. 97

Scott Sanner and Sungwook Yoon. International Probabilistic Planning Competition (IPPC). http://users.cecs.anu.edu.au/s̃sanner/IPPC_2011/, 2011. 97

Paul Scerri, David V. Pynadath, and Milind Tambe. Why the elf acted autonomously: Towards a theory of adjustable autonomy. In *The First International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2002, July 15-19, 2002, Bologna, Italy, Proceedings*, pages 857–864. ACM, 2002. 2, 107

Sven R Schmidt-Rohr, Steffen Knoop, Martin Lösch, and Rüdiger Dillmann. Reasoning for a multi-modal service robot considering uncertainty in human-robot interaction. In *Proceedings of the 3rd ACM/IEEE International Conference on Human Robot Interaction*, pages 249–254. IEEE, 2008. 5, 63

Guy Shani and Ronen I Brafman. Replanning in domains with partial information and sensing actions. In *Proceedings of the Twenty-Second international Joint Conference on Artificial Intelligence*, pages 2021–2026. AAAI Press, 2011. 115

Guy Shani, Joelle Pineau, and Robert Kaplow. A survey of point-based POMDP solvers. *Autonomous Agents and Multi-Agent Systems*, 27(1), 2013. 19

Thomas B Sheridan. *Telerobotics, automation, and human supervisory control*. MIT press, 1992. 109

Marcel Steinmetz, Jorg Hoffmann, and Olivier Buffet. Revisiting Goal Probability Analysis in Probabilistic Planning. In *Proceedings of the 26th International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI Press, 2016. 82, 83, 89, 90

Florent Teichteil-Königsbuch. Stochastic safest and shortest path problems. In *Proceedings of the NCAI*, 2012. 3, 87

F. Trevizan, F. Teichteil-Königsbuch, and S. Thiébaux. Efficient Solutions for Stochastic Shortest Path Problems with Dead Ends. In *Proceedings of 33rd Conf. on UAI*, 2017. 3, 4, 82, 83, 86, 87, 90

Hans Van Ditmarsch, Wiebe van Der Hoek, and Barteld Kooi. *Dynamic epistemic logic*, volume 337. Springer Science & Business Media, 2007. 40, 117, 118, 119

Manuela Veloso, Joydeep Biswas, Brian Coltin, and Stephanie Rosenthal. Cobots: Robust symbiotic autonomous mobile service robots. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4423–4429, 2015. 2, 6, 63, 64

K. Yamazaki, R. Ueda, S. Nozawa, M. Kojima, K. Okada, K. Matsumoto, M. Ishikawa, I. Shimoyama, and M. Inaba. Home-assistant robot for an aging society. *Proceedings of the IEEE*, 100 (8), 2012. 1

Kimitoshi Yamazaki, Ryohei Ueda, Shunichi Nozawa, Yuto Mori, Toshiaki Maki, Naotaka Hatao, Kei Okada, and Masayuki Inaba. Tidying and cleaning rooms using a daily assistive robot. *Paladyn*, 1(4), 2010. ISSN 2081-4836. 1

H. A. Yanco and J. Drury. "Where am I?" Acquiring situation awareness using a remote robot platform. In *Proceedings of the 2004 IEEE International Conference on Systems, Man and Cybernetics*, volume 3, Oct 2004. 6

Håkan LS Younes and Michael L Littman. PPDDL1. 0: An extension to PDDL for expressing planning domains with probabilistic effects. *Techn. Rep. CMU-CS-04-162*, 2004. 2, 99