Plan Recognition as Planning

Miquel Ramírez

TESI DOCTORAL UPF / 2012

Thesis supervisor

Prof. Dr. Héctor Geffner,
ICREA
Department of Information and Communication Technologies

UNIVERSITAT POMPEU FABRA

The PhD thesis committee was appointed by the rector of the Universitat Pompeu Fabra on ...., 2012.

Chairman

Member

Member

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

CHAIRMAN	MEMBERS

To my parents, Francisco Ramírez and María Jávega.

Acknowledgements

First and foremost, I want to mention my fellow PhD students, in no particular order, Emil Keyder, Nir Lipovetzky and Alexandre Albore. I do not think that Hector could have chosen better students, nor I could have been more lucky than to share this amazing journey with them. Thanks for the many discussions, which I hope were equally enlightening to all the parties involved, as well as for the mutual assistance we have given to each other over the years. Thanks as well for the many parties and long nights we have shared over those same years. I will be really missing you punks.

I want to thank to Blai Bonet and Patrik Haslum for their codes and many interesting, clear and thought–provoking papers. I learnt from both quite a few things that have proven to be invaluable during my PhD. Also I learnt from them how to write a planner from scratch, one that *actually* worked and was workable after the deadlines.

Next I want to mention the never ending and tireless support provided by Lydia García and her fellow DTIC administration workers. Without her – their – devotion, the task of completing this PhD would have been considerably harder. Thanks for dealing on my behalf with those trivial "real world" things such as contracts, expenses from travels and in general, the startlingly copious paper trail that research leaves in its wake.

Now I want to thank my friends Candela, Iñigo, Sandra and Eider, as well as my brothers Francisco and Joan Manel for the patience they have had while coping with me. When I did not return calls at all. Or when I was very late answering e-mails. Or just answered no, perhaps too quickly, to any proposal to do something which can fit into that notion of *having a life of your own.* Thank you a lot for coping with the apparent indifference. If something is hard when doing research is *precisely* to not forget about having a life of your own.

Just before finishing I want to thank my parents Francisco and Maria for the never faltering support and unconditional love they have given to me since I was born. They are ones responsible of my nearly insatiable curiosity and tireless will to learn new things. I also want to thank them for the healthy skepticism they have shown in the last six years about this "academia" thing. They have been the "voices of reason", which even when completely disregarded, are useful reminders about worldly affairs. Perhaps the academia might not pay as well as you would wish for one of your sons, but I do really think it is a worthy enterprise. Rest assured that I will never – I cannot – forget where I come from, and I come from you.

I reserve the last slot for Héctor Geffner, my tutor and the mastermind behind this thesis. I expect to live up to the fine example he sets by being generous enough to provide his students with directions and topics to pursue, his efforts to get us "to push the envelope” so we make our best at all times. My PhD experience can be summarized in the following questions and answers:

- Need a pointer? Héctor knows the right one.
- Want something interesting to read? Ask Héctor.
- Wondering about what is best to do next? Knock on his door and ask for advice.

If there is anything like a “Best Tutor Award” in some AI venue, you should be the one getting it next year. Thank you Héctor for everything, the thing you have done for me is priceless. Something I will not be able to pay you back other than doing the same myself for any PhD students I have in the future.

Abstract

Plan recognition is the problem of inferring the goals and plans of an agent after partially observing its behavior. This is the inverse of planning, the problem of finding the actions that need to be done in order to achieve a goal. In this thesis we show how the problem of plan recognition can be solved using unmodified, state–of–the-art planning algorithms and representation languages. Along with a solid computational framework for deriving posterior goal probabilities, we introduce a novel and crisp model–based formulation of plan recognition whose flexibility surpasses that of previous approaches.

Resum

Plan recognition és un problema computacional que consisteix en identificar el propòsit d'un agent intel·ligent, havent observat parcialment el seu comportament. Aquest és el problema invers al problema de la planificació automàtica, que consisteix en trobar les accions que són necessàries dur a terme per tal d'aconseguir un cert objectiu. En aquesta tesi mostrem com el problema de *plan recognition* és pot resoldre mitjançant els mateixos algorismes i llenguatges de representació utilitzats per resoldre la planificació automàtica. La nostra proposta no només contè una serie d'algoritmes eficients i robustos, sinò que ve suportada per un marc teòric formal, que ofereixen en conjunt una flexibilitat de la qual no disposen propostes prèvies per resoldre *plan recognition*.

Resumen

Plan recognition es el problema computacional que consiste en deducir el objetivo perseguido por un agente inteligente tras haber observado de forma incompleta su comportamiento. Éste es el problema inverso al problema que plantea la planificación automática, que consiste en encontrar las acciones que se necesitan llevar a cabo para conseguir un cierto objetivo. En esta tesis mostramos como el problema de *plan recognition* se puede resolver mediante los mismos algoritmos y lenguajes de representación utilizados resolver la planificación automática. Nuestra propuesta no tan sólo ofrece una serie de algoritmos eficientes y robustos, sino que además viene apoyada por un marco teórico formal, que ofrecen, en su conjunto, una flexibilidad de la cual carecen propuestas anteriores para resolver el problema de *plan recognition*.

Preface

> To a man with a hammer, everything looks like a nail.
>
> Mark Twain

The problem of plan recognition has been for a long time an important topic in Artificial Intelligence research. Solving plan recognition problems is essential for autonomous systems that need to make sense out of the observed behavior of humans. That is, given a set of *possible goals* and a *sequence of actions* done by an agent to achieve one of these goals, select what goal accounts best for the observed actions.

Actions can represent many different types of events, depending on the application. Sequences of utterances are relevant for Natural Language Processing applications dealing with automated dialogue or synthetic advice. For automated surveillance, cognitive assistance for the elderly or home automation applications, traces of goal–oriented behavior are sought on readings obtained from sensors such as cameras, GPS systems or RFID scanners, tracking changes in location of people and objects over time. Intelligent user interfaces try to anticipate user actions by analyzing logs of events generated by a user interacting with a shell or a graphical user interface.

As an illustration, let's consider the following example of a plan recognition problem, adapted from (Schmidt et al., 1978), where the observed actions are specified as follows

1. O_1: Steve walked to the cabinet.
2. O_2: He opened the cabinet.
3. O_3: He took a record out of the cabinet.
4. O_4: He took the record out of the record jacket.

The possible goals that might Steve be pursuing are

1. G_1: Steve wants to prepare his breakfast.
2. G_2: Steve wants to listen to a record.

While G_2 looks to us as the obvious answer, a computer system would need two things to reach the same conclusion we have. First, it needs the knowledge relevant and essential to establish a causal connection between actions and goals. Second, some means to process this knowledge to conclude that G_2 is a better explanation than G_1.

This knowledge would include a description of the actions the agent may do, portraying the *causal relationships* between changes in the agent environment and the actions it can do, such as "in order to listen a record, the agent needs to put the record on the player turntable and turn the player on", as well as constraints like "a record needs to be out of the jacket to be put on a turntable". Complementing this factual knowledge there may be generic a set of assumptions on the constraints the agent takes into account during the process of deciding what actions to do. For instance, it can be assumed that the agent will prefer to achieve its goal in the most efficient manner possible, that is, in a *rational*, or optimizing, way. This notion of "rationality" is *bounded* by constraints which might impair agent performance, such as limited knowledge about its environment or knowledge about how to act optimally. In the example above, Steve might not know where the record he wants to listen is, so he might need to search for it. This would require to invest more time or energy to achieve its goal than when knowing precisely the record whereabouts.

With the previous assumption in place, ranking the possible goals with respect to how well do they account for the observed behavior, involves computing the likelihood of the observed actions assuming that each goal is the one the agent is actually seeking. Observed actions likelihood can be decomposed into two independent factors. Sequences of actions found to be *consistent* with the facts and assumptions provided are considered to be more likely, all other things being equal, than sequences of actions which are not. When observations are found to be consistent with the assumed goal and the causal theory of agent behavior, then it is required to quantify how *necessary* they are. This amounts to compare the number of plans consistent with the theory and the assumed goal that are *not* consistent with the observed sequence of actions, with the number of such plans which are *consistent* with the observed actions.

Existing approaches to plan recognition are best understood by studying how the range of possible agent behaviors is represented – causal theories of agent behavior, assumptions made on agent knowledge of its environment and its criteria to prefer one course of action over another – and how they rank or select possible goals depending on how well these support the observed actions.

Our approach to the challenges posed by plan recognition relies on the general idea that plan recognition is automated planning *in reverse*. While automated planning is the problem of seeking a sequence actions that accounts for a given goal, plan recognition is the problem of seeking the goals that account for the sequence of actions observed. We claim that our approach is the *first* that seriously formulates *and* exploits computationally the connection, first described in (Schmidt et al., 1978), between goal–oriented behavior generation and plan recognition.

The representation and computation aspects of our approach are built on top of the–state–of–the–art of automated planning modeling languages and solvers. Causal theories of behavior are expressed in the language of *Functional* STRIPS, which is simple yet expressive enough to account for a broad variety of agent cognitive and physical abilities as well as characterizing the environments where the agent performs. Goal likelihood computation addresses both the *consistency* and *necessity* of observed action sequences by invoking automated planners in an off–the–shelf manner. While our contribution shares with earlier works such as (Schmidt et al., 1978; Lesh and Etzioni, 1995; Baker et al., 2009) the "plan recognition is planning in reverse" approach to plan recognition, we consider it to provide a general yet crisp formal framework built on solid, efficient computational foundations, something which previous approaches have delivered with limited success.

The significance of our contribution to plan recognition has been appreciated by the AI community. Parts of the present work have been published in:

- "Plan Recognition as Planning" by M. Ramirez and H. Geffner, published at the 22nd International Joint Conference on Artificial Intelligence (Ramirez and Geffner, 2009).
- "Probabilistic Plan Recognition as Planning" by M. Ramirez and H. Geffner, published at the 24th Annual Conference of the Association for the Advancement of Artificial Intelligence (Ramirez and Geffner, 2010).
- "Goal Recognition over POMDPs" by M. Ramirez and H. Geffner published at the 23rd International Joint Conference on Artificial Intelligence (Ramirez and Geffner, 2011).

Contents

List of Figures

List of Tables

Part I

Planning Background

Chapter 1

Introduction

1.1 Motivation

The need to recognize the goals and plans of an agent from observations of his behavior arises in a number of applications in the fields of natural language processing, multi-agent systems, and assisted cognition (Cohen et al., 1981; Pentney et al., 2006; Yang, 2009). This automated reasoning task is referred to as the *plan recognition* problem (Cohen et al., 1981; Pentney et al., 2006; Yang, 2009) and has been addressed using a variety of methods which include specialized procedures (Kautz and Allen, 1986a; Avrahami-Zilberbrand and Kaminka, 2005; Lesh and Etzioni, 1995; Huber et al., 1994), parsing algorithms (Pynadath and Wellman, 2000; Geib and Goldman, 2009) and inference procedures on Bayesian networks (Charniak and Goldman, 1993; Bui, 2003).

Plan recognition can be best understood as planning in reverse: while in planning the goal is given and a plan is sought; in plan recognition, part of a plan is given, and the goal and complete plan are sought. Early approaches to plan recognition (Schmidt et al., 1978; Perrault and Allen, 1980) adopted this view. The space of possible behaviors was described *implicitly* as a *planning problem* and goals were selected when a plan that fitted the observations and achieved the goal was found.

This approach was in general abandoned from the late 1980s on. Since then, in most approaches to plan recognition the space of possible plans or activities to be recognized is assumed to be a fixed set of stereotypical plans (Kautz, 1991). These are then compiled into some suitably defined *library* of plans or policies by the plan recognition system designer. Plan recognition thus became a problem of *matching* observations with a fixed set of plans, a *plan library*, stored in some suitable hierarchical representation similar to And/Or graphs or Htn *task networks*.

The reasons for abandoning the plan recognition as planning approach were two. First, a fundamental weakness was found in early formulations. Showing that a plan fitting the observations exists for some goal is insufficient to rule out other goals as not possible. There may exist many and indeed an infinite number of plans that comply with the observations, yet some of them, under certain assumptions, are much more plausible than others. Second, checking that such plans exist requires solving a *search problem* over the space of plans that achieve the hypothesized goals.

The size of this search problem grows with the amount of detail used to model agent actions and environment. Unfortunately, the domain–independent planners available at the time were unable to scale up well, or at all, over planning domains modeling complex agent behavior (Kautz, 1991).

Over the years the planning community has caught up with the challenges posed by the combinatorial explosion in plan search, pushing the envelope of feasible problems by developing planners with increased capabilities that include

1. 1995 – GraphPlan (Blum and Furst, 1995)
2. 1999 – BlackBox (Kautz and Selman, 1999)
3. 2000 – Hsp (Bonet and Geffner, 2001a)
4. 2001 – Ff (Hoffmann and Nebel, 2001)
5. 2006 – Fast Downward (Helmert, 2006)
6. 2008 – Lama (Richter and Westphal, 2008)

The once upon a time valid concerns regarding scalability seem to no longer apply, and we consider that the work done by the automated planning community can once more contribute to advance the state–of–the–art in plan recognition.

The goal of this thesis is to provide a model–based account of plan recognition, which overcomes the limitations regarding the formulations and computational models discussed in previous works relating plan recognition and planning. We want our formulation to provide well–founded principles which use the properties of plans achieving a goal in order to deem the goals as a good explanation of the observations. The computational model will be relying on existing planners without requiring modifications, so its scalability is directly related to that of the planner being used.

The advantages to our approach are several. A model–based approach is much more flexible and robust than a library–based approach since the system designer does not have to anticipate the plans that the agent might use to achieve a goal. This rules out the possibility that some goal is not selected as an explanation of the observed actions because the plan the agent actually used was not in the library. Furthermore, planning languages empower modelers so they can construct rich agent models in a intuitive and straightforward way. For instance, to encode complex goals in a succint way, implicitly denoting a vast number of world states.

Our approach is not without disadvantages, though. Combinatorial explosion in plan search is a moving target both for the planning community and plan recognition system developers, since the latter will eventually come up with an agent model on which no state–of–the–art planner scales well. This is particularly of concern for agent models considering non–deterministic action effects and partial–feedback, since the planners available for such models are not, relatively speaking, as powerful as planners for less expressive agent models.

1.2 Example: RoboSchool

Let us consider the following example plan recognition task. The observing agent is a robot, which is required to figure out the goal being sought by a human trainer, so it can imitate later the trainer's behavior. We assume that both the robot and the trainer have *perfect* knowledge of their environment i.e. know exactly their own position and that of objects relevant to accomplish the task. We will also make the assumption that the actions the trainer does are *deterministic*, that is, they have always the same effect and she receives immediate and complete feedback about their effects.

The environment where the activity takes place is a lab whose layout presents some constraints on the trainer movement. The lab itself is modeled as discrete, square cells which might be blocked by furniture. The robot is able to accurately observe changes in the location of the trainer and assumes it can move from any unblocked cell to any of its at most eight adjacent unblocked cells. These are taken to be to be *move* actions, allowing the trainer's motion to be modeled as a set of discrete actions.

The objects of interest in this task are of two kinds. One is a set of wooden pieces – denoted p_1, p_2, ..., p_n – which might have one out of three shapes – cube, cone or sphere – and one out of three possible colors – red, blue and green – which lay scattered on the lab floor. The other is a set of bins – denoted b_1, b_2, ..., b_n – that can hold a number of wooden pieces much larger than the ones present in the lab. The bins are located in cells, either accessible or inaccessible, as long as the trainer can get adjacent to those cells. An example initial situation is depicted in Figure 1.1.

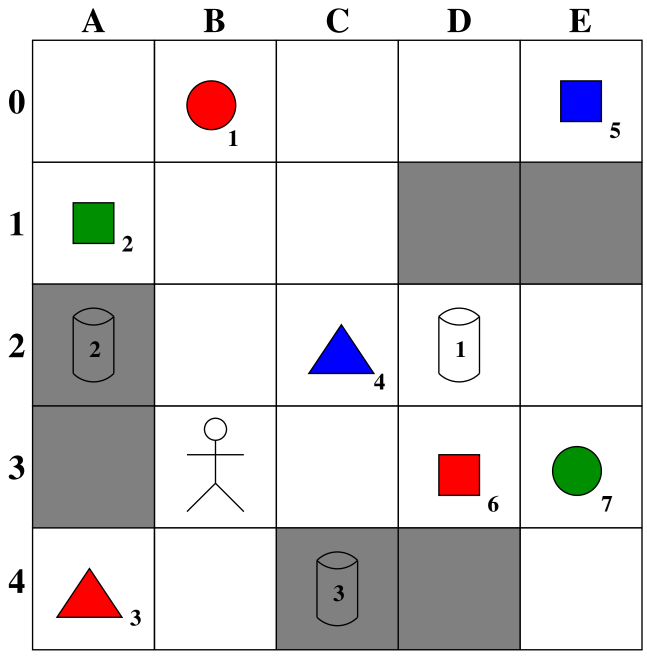

Figure 1.1: Lab environment represented as a grid, each cell is referred to by its coordinates: $A0$, ..., $E4$. Clear cells are the ones the trainer can enter and exit. Numbered cylinders 1, 2 and 3 represent bins b_1, b_2 and b_3. Note that b_1 is located in a cell the trainer cannot traverse. Objects are represented by their shapes and colors. The red circle is denoted as p_1 and the blue triangle as p_4. The trainer is initially located in cell $B3$.

Besides their shape and color, objects are further described by their situation: lying on the ground, being carried by the trainer or stored inside one particular bin. Object situation is assumed to change only when the trainer does some action, which can only be executed when a specific condition is met.

The trainer is expected to *pick* up the objects from the ground and to store them in some bin. The trainer can only pick those objects which lie in the same cell she is standing in, and it is guaranteed that she will be carrying it after the action is done. She can *drop* them – provided that she already carries them – into bins in the same cell or *throw* them into those bins in an adjacent cell. In both cases it is guaranteed that the object will be stored in the appropriate bin after the action is done. The trainer can carry an unlimited number of objects simultaneously and the bins can also store an unlimited number of objects.

The set of possible goals that the robot knows about and expects the trainer to pursue are:

1. Store all triangles in b_1.
2. Store all spheres in b_2.
3. Store all cubes in b_3.
4. Store red objects in b_2.
5. Store green objects in b_3.
6. Store blue objects in b_1.

One possible *plan* for the trainer when she is pursuing task #1, given that she starts at cell $B3$, would be:

1. Walk from $B3$ into $A4$.
2. Pick p_3 up.
3. Walk from $A4$ into $B3$.
4. Walk from $B3$ into $C2$.
5. Pick p_4 up.
6. Throw p_3 into b_1.
7. Throw p_4 into b_1.

Unfortunately, robot sensors are far from being 100% accurate and tend to miss certain actions with some probability. For instance, the robot can entirely miss up to 70% of *walk* actions and up to half of either *pick* or *drop* actions. Hence, a possible sequence of actions that the robot may observe is:

1. Pick p_3 up.
2. Walk from $A4$ into $B3$.

Under certain assumptions, it can be deduced that tasks #1 and #4 are equally *likely* and good explanations for the observations obtained, while the other four tasks are – to a varying degree – rather *unlikely* explanations for the events observed. Namely, if the robot is assuming that the trainer tries to attain a goal in the *most efficient* manner, it would make little sense that she picked up the red triangle if she is storing all spheres in b_2, or storing all green objects in bin b_3.

While this task looks to be rather simple, it is not far from the degree of sophistication of current robotic platforms such as SIMON [1], developed by Georgia Tech's Socially Intelligent Machines Lab, and discussed in recent work on *social learning* and multi–agent systems (Knox and Stone, 2010).

1.3 Planning and Plan Recognition

The "generative", or model–based, approach to plan recognition formulated in this thesis has several advantages over approaches to plan recognition based on matching observations against plans stored in a plan library.

First, it does not require a library of plans but a *domain theory* from which plans can be automatically constructed. This offers a more general and flexible framework than the one offered by library–based approaches. While it is known that propositional, acyclic plan libraries can be efficiently compiled into domain theories (Lekavý and Návrat, 2007), the converse is not true. Because of this, behavior unforeseen by the plan recognition system designer can easily be unaccounted for. Solving this problem of incompleteness by means of systematically enumerating possible plans implies a combinatorial blow up in the size of the plan library, defeating parsing and graph–search algorithms that offer polynomial run–time guarantees over the *size* of the library. Methods based on obtaining observed actions likelihood from plan libraries encoded into Dynamic Bayesian Networks (Bui, 2003) ameliorate this problem, but only if the *length* of possible plans is *bounded* by a constant.

It might be contended that the claim above is of little practical use since domain theories are not necessarily easier to come up with than plan or policy libraries for anything more complex than the "toy", "grid–based", task described in the previous section. This might be true for applications dealing with plan recognition tasks where there is no clear discrete notion of agency. However, we argue that such applications are few, or in an early stage of development. Coming up with domain theories for applications dealing with designing highly structured processes, such as the internal business processes of a company, can be trivially achieved by retrieving the specification and internal state of the graphical user interface (Hoffmann et al., 2010) component in a business management application. The automated generation of domain theories, in order to compute plans in an on–line way, has been shown to be easily achieved even in applications rarely discussed in AI research such as real–time video games (Orkin, 2005).

Second, a major shortcoming of library–based approaches lies in using a language, either graphical or based on production rules, with very limited expressiveness when it comes to characterizing world states (Geib and Goldman, 2009). One example are the short–comings such approaches have when dealing with plans that achieve several different goals simultaneously. Automated planning domain theories use a factored representation of world states (Fikes and Nilsson, 1971; Geffner, 2000), where particular states are described as *logical formulas* that model one or several possible world state variables valuations. Model–based plan recognition approaches identify processes *indirectly* rather than recognizing them *directly*, analyzing the *outcomes* of plan execution and the *constraints* posed by world state variables values on action execution.

[1] SIMON was recently featured in New York Times, see `http://www.athomaz.com/?p=449`

To illustrate this point, consider adding the goal "store all triangles and spheres in b_1" to the example presented in Section 1.2. To model that goal is as simple as writing the logical formula describing world states where all objects with those specific shapes are placed inside bin b_1, according to the syntax and grammatical rules of the domain–independent planning language used. This is far more natural and elegant than ad–hoc solutions such as the ones proposed by (Geib and Goldman, 2009).

The third advantage, is that it becomes possible to build plan recognition systems on top of state–of–the–art planners, guaranteeing that they scale as well as planners do. As discussed in Section 1.1, the concern about planners not being able to handle the combinatorial explosion in the size of the set of behaviors entailed by a domain theory, no longer holds. The challenge, though, is to come up with a domain theory expressive enough to capture all relevant aspects of the plan recognition task, and at the same time, structured in such a way that the domain–independent heuristics driving the search procedures used by planners remain effective. This challenge is not particular to planning, and has been the subject of extensive discussion in the SAT solving community (Walsh, 2000; Fey et al., 2006; Hoos, 1999), where powerful domain–independent solvers can be defeated with some particular encodings of an input problem, while dealing effortlessly with a different encoding of the same problem.

1.4 Overview of Contributions

The main contribution of this thesis is a novel and crisp formulation of *single–agent*, *keyhole* (Kautz and Allen, 1986a) plan recognition as planning where *posterior goal probabilities* $P(G|O)$ (Charniak and Goldman, 1993) follow from the costs of plans. Goals G are *logical formulas* over variables describing *many world states* simultaneously. Observation sequences O are *totally ordered* sequences of *action signatures* expressed in a planning language, which encode parts of the plan executed by the observed agent. The formulation does *not* require O to be a plan *prefix*, that is, the first n actions in a plan, but rather handles *any subset* of actions of a plan, as long as these are ordered without placing any requirement on the number of actions n observed.

To produce the posterior probabilities $P(G|O)$, a prior distribution $P(G)$ over the goals G is *assumed* to be given. The likelihoods $P(O|G)$ of the observation O given the goals G are defined in terms of the *cost differences* between plans π that achieve G and have O embedded, and those that still achieve G but do not comply with O. Plans π are computed by a *classical* planner over a classical domain theory. The planners are used off–the–shelf so they do not need to be modified in any way. The domain theory is assumed to be shared between agent and observer except for the true goal of the agent that is hidden to the observer.

We also show how to extend the model-based approach above to plan recognition over POMDP domain theories, where actions are stochastic and states are partially observable. While the definition of $P(G|O)$ and $P(G)$ is unchanged, $P(O|G)$ is computed indirectly from the value function $V_G(b)$ over beliefs b generated by a POMDP planner for each possible goal G. Once computed, executions are sampled from this value function, assuming that the agent tends to select those actions that are likely to achieve the goal more cheaply. The likelihood of the observations O given the goal G is approximated from these samples.

In analogy to the other cases, the goal recognition problem over POMDPs is solved using an off-the-shelf POMDP planner. While POMDP planners do not scale up as well as MDP planners, and certainly much worse than classical planners, we show that still a rich variety of recognition problems involving incomplete information can be effectively modeled and solved in this manner.

In all cases, the expressive power and computational feasibility of the approach will be illustrated through a number of experiments over several domains.

1.5 Outline of the Thesis

In Chapters 2 and 3 we cover the relevant background on Classical, MDP and POMDP planning relevant to this thesis. We start by giving a formal account of the structure and semantics of planning problems, as well as the algorithms available to solve them. We will illustrate the expressiveness allowed by planning languages, by showing the reader how the example presented in Section 1.2 can be encoded.

In Chapter 4 we will present the formulation of plan recognition over classical domain theories, formally defining plan recognition problems and their solution, in terms of costs of classical plans. We will then discuss in detail the domains used to evaluate our approach, so the reader can further appreciate the challenge and significance of each of the domains. We end this chapter presenting our empirical results and discussing their implications.

The extension of the formulation presented in Chapter 4 to domain theories modeling partially–observable, stochastic environments is presented in Chapter 5. As in the previous chapter, formal definitions will be given for both the problem and the algorithms to solve them, followed by a detailed discussion of the evaluation domains and the obtained empirical results.

A selection of the most relevant previous approaches to plan recognition are reviewed in Chapter 6, as well as those works which have had a direct influence on the contributions presented in this thesis.

We conclude with Chapter 7 offering a summary of the main contributions and empirical results. We will explain their significance and implications for both plan recognition and planning. We will also offer an discussion of existing limitations in the approach along with proposals to further extend its applicability and scalability.

Chapter 2

Classical Planning

In this chapter we present the classical planning model, and discuss how it can be described implicitly with factored representations. We then briefly review some complexity results of finding solutions for classical planning models, which while in theory (Bylander, 1994) are quite hard to solve, in practice such solutions can often be obtained with ease (Helmert, 2005). Most of classical planning systems, or *planners*, which have come out to become winners of the International Planning Competitions, on the basis of the number of problems they solve, how fast these are solved, and the quality of the solutions found[1], are based on the "planning as heuristic search" approach. We will discuss briefly what these *heuristics* are in this context and introduce the search algorithms which are the backbone of the planners we use in Chapter 4. After that, we will give a detailed overview of the h^m family of heuristic estimators, the additive heuristic h_{add}, relaxed plan heuristics and the *landmark* heuristics. As a whole these constitute the bulk of the heuristic machinery that state–of–the–art classical planners derive *automatically* from factored representations of planning problems in order to search effectively for plans. We will end this chapter with a complete overview of the Strips representation of the example given in Section 1.2.

2.1 The Classical Planning Model

Classical planning is the problem of finding a sequence of actions, or *plan*, that when applied in the initial state of the problem results in a goal state. A classical planning problem can be viewed as a directed graph (Nilsson, 1980; Pearl, 1983; Newell and Simon, 1972) whose nodes represent states of the planning problem, and whose edges represent state transitions caused by actions. Edges' source nodes are the state in which actions are performed, and target nodes correspond to resulting states. A plan is then a path from the node in the graph representing the initial state to a node representing a state that is an element of the set of goal states of the problem. The formal *state–space model* (Hart et al., 1968; Nilsson, 1980) underlying the planning problem can be described as follows:

[1]Only considered for non–optimal planners.

Definition 2.1 (Classical planning model)**.** *A planning model* $\Pi = \langle S,\ s_0,\ S_G,\ A,$ $app,\ f\rangle$ *consists of:*

- *A* finite *and* discrete *set of states* S*,*
- *An* initial state $s_0 \in S$*,*
- *A set of* goal states $S_G \subseteq S$*,*
- *A set of* actions A*,*
- *The* applicability *function* $app : S \to \mathcal{P}(A)$ *giving the set of actions that are applicable in each state,*
- *The* transition *function* $f : S \times O \to S$*, where* $f(s, a)$ *is defined only when* $a \in app(s)$*.*

The state resulting from applying an action a in a given state s is $f(s, a)$, denoted also as $s[a]$. The result of applying a sequence of actions $\pi = (a_1, \ldots, a_n)$ to state s can be defined recursively as

$$\begin{aligned} s[\epsilon] &= s \\ s[a] &= \begin{cases} \bot & \text{if } a \notin app(s) \\ f(s,a) & \text{if } a \in app(s) \end{cases} \\ s[a_1, \ldots, a_n] &= (s[a_1, \ldots, a_{n-1}])[a_n] \end{aligned}$$

Sequences of actions that solve the planning problem, mapping the initial state s_0 into some state $s \in S_G$ are *plans*:

Definition 2.2 (Plan)**.** *Given a planning model* Π*, a* plan π *for* π *is a sequence of actions such that* $s_0[\pi] \in S_G$*. The* length π *of a plan is the number of actions in a plan.*

In the absence of a cost function, plans with lower length are preferred as solutions than those with higher length. Alternatively, the planning model may be stated in conjunction with a *cost function* that assigns costs to the actions of the problem:

Definition 2.3 (Classical planning with costs)**.** *A planning model with costs,* Π_c*, consists of a planning model* $\Pi = \langle S, s_0, S_G, A, app, f\rangle$ *along with a function* $c : A \to \mathbb{R}_0^+$ *that maps each action in the model to a non–negative cost.*

Such costs are considered to be *additive*, making the definition of cost of a plan straightforward:

Definition 2.4 (Plan cost)**.** *The cost of a plan* $\pi = (a_1, \ldots, a_n)$ *is given by*

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

In planning problems with costs, plans with lower cost are preferred over plans with higher cost. Planning without cost, in which length $|\pi|$ is the criteria for preference, can be seen as an special case of planning with costs in which $c(a) = 1$ for all $a \in A$.

2.2 Factored Representations in Strips

Since it is impractical to explicitly enumerate the state space of planning problems large enough to be considered of interest, *factored* representations, in which states are represented by complete assignments to a set of *variables* with *finite* and *discrete* domains, are commonly used. Given such a representation, the sets of goal states S_G and operators, as well as the applicability $A(s)$ and transition $f(s, a)$ functions, can also be described in terms of these variables.

The most common and simplest factored representation is one which consists of only boolean variables, with the value of each such variable indicating whether a proposition about the world is true or false at a given state. Such variables are also known as *fluents* or *facts.* This representation was first used in the Stanford Research Institute Problem Solver (STRIPS) (Fikes and Nilsson, 1971) and is known by that name today.

The STRIPS language comprises two parts: a language for describing the world, the set F of fluents, and a language for describing how the world changes, the set of actions A, a compact representation of actions that avoids the frame problem (McCarthy and Hayes, 1969). We will consider the STRIPS language as used currently in planning, i.e. the STRIPS subset of PDDL (McDermott, 2000), rather than the original version of STRIPS which is slightly more complex (Fikes and Nilsson, 1971; Lifschitz, 1986) and was used in previous approaches to plan recognition (Schmidt et al., 1978; Perrault and Allen, 1980).

The fluent set F is made up of expressions obtained from two types of symbols: *relational* and *constant* symbols. For instance, fluents accounting for one of the possible trainer locations in the example described in Section 1.2, would be

$$atTrainer(A0)$$

where *at* is a relational symbol of arity 1, denoting the location of the trainer, and cell coordinates $A0$, ..., $E4$ are constant symbols. Fluents are defined in a standard way from the combination $p(t_1, \ldots, t_k)$ of a relational symbol p and a tuple of terms t_i of the same arity as p. STRIPS *formulas* are obtained by closing the set of fluents under the conjunctive ($\wedge$) propositional connective, and are identified with sets of atoms.

A planning problem described in the STRIPS language, is defined as follows:

Definition 2.5 (STRIPS)**.** *A planning problem in* STRIPS*, $P = \langle F, A, I, G\rangle$ consists of:*

- *A set of boolean variables (fluents) F,*
- *A set of tuples A representing actions, each having the form $a = \langle Pre(a)$, $Add(a)$, $Del(a)\rangle$, where $Pre(a)$, $Add(a)$, $Del(a)$ are all in F and describe action precondition and effects,*
- *A set $I \subseteq F$, describing the initial state,*
- *A set $G \subseteq F$, describing the set of goal states.*

We note that many planning problems may share a common structure, such as initial state, fluent and action sets. We will refer to this base structure as a planning *domain*. We note that this term can be found in the existing planning literature to refer to the object described in Definition 2.5, or also as the fluent and action sets shared by many planning problems. In the context of the present thesis, a planning *domain* refers to the following

Definition 2.6 (STRIPS Planning Domain). *A planning* domain $P[\cdot] = \langle F, A, I \rangle$ *consists of:*

- *A set of boolean variables (fluents) F,*
- *A set of tuples A representing actions, each having the form $\langle Pre(a)$, $Add(a)$, $Del(a)\rangle$, where $Pre(a)$, $Add(a)$ and $Del(a) \subseteq F$.*
- *A set $I \subseteq F$, describing the initial state.*

Planning problems can be then obtained by attaching to a domain description $P[\cdot]$ a goal conjunction $G \subseteq F$, resulting in problem $P[G] = \langle F, I, A, G \rangle$.

A significant difference between relational and constant symbols in STRIPS is that the former are used to keep track of aspects of the world that may change as result of the actions – the current location of the trainer – while the latter are used to refer to objects in the domain – discrete locations – which are not affected in any way by action effects. This *dynamic* nature is the main reason for referring to the expressions describing an aspect of the world state as *fluents* rather than by the standard term in logic, *atom*.

Actions $a \in A$ are defined over the set of fluents F, by specifying their *precondition*, *add*, and *delete* sets (conjunctions) of fluents in F, $Pre(a)$, $Add(a)$ and $Del(a)$. Actions are normally defined by means of first–order *schemas*, as is indeed the case in the STRIPS subset of PDDL. In this thesis we assume that such schemas have been previously grounded.

A STRIPS planning problem $P = \langle F, A, I, G \rangle$ is given a precise meaning by mapping it into the planning state model $\Pi = \langle S, s_0, S_G, A, app, f \rangle$ where:

1. States S are sets (conjunctions) of fluents in F,
2. The initial state s_0 corresponds with I,
3. The goal states S_G are those states $s \in S$ such that $G \subseteq s$,
4. The applicability function $app(s)$ maps states s into sets of actions such that
$$app(s) = \{a \mid Pre(a) \subseteq s,\ a \in A\}$$
5. The transition function $f(a, s)$, $a \in A$, $s \in S$ is such that
$$f(a, s) = s \cup Add(a) \setminus Del(a) \tag{2.1}$$

The solution of the planning problem P is the solution of the state model Π, namely, a sequence of applicable actions that map the initial state into goal state in Π:

Definition 2.7 (Valid Plan). *A solution of the planning problem $P = \langle F, A, I, G \rangle$, or* valid plan*, is a sequence of actions $\pi = (a_1, \ldots, a_n)$ such that*

1. $Pre(a_1) \subseteq I$,
2. $Pre(a_i) \subseteq I[a_1, \ldots, a_{i-1}]$ *for* $i > 1$,
3. $G \subseteq I[a_1, \ldots, a_n]$.

The set of solutions, or *valid plans* to a planning problem $P[G]$ is denoted as $\Pi_P(G)$.

The extensions of Strips described in the Pddl specification (McDermott, 2000; Fox and Long, 2003) such as those involving negated fluents, *conditional effects* (Anderson et al., 1998), disjunctive and quantified (existential or universal) precondition and effect formulas are easily compiled into the present formulation of Strips (Gazen and Knoblock, 1997). Most, if not all, state–of–the–art planners since ff (Hoffmann and Nebel, 2001) provide implementations of such compilations integrated with efficient routines for grounding Pddl action schemas.

Conditional Effects

We will stop at examining one of the extensions above mentioned, that of *conditional effects*. This extension is fundamental to the formulation of plan recognition as planning given in Chapter 4.

As their name imply, conditional effects extend the expressiveness of STRIPS actions a by allowing to define any number of effects e which only affect the state s' resulting from executing a on a state s, when a certain condition c is met by s.

We will refer to the i-th conditional effect associated to action a as $ce(a)_i$:

$$ce(a)_i : c_i \rightarrow ca_i, cd_i$$

where c_i, a_i and d_i are all subsets of fluents in F. c_i is the *condition* which is required to be true in state s in order to trigger the effects ca_i and cd_i.

The *transition function* $f(a, s)$ in Equation 2.1 is now defined as

$$f(a, s) = s \cup Adds(s) \setminus Dels(s)$$

where $Adds(s, a)$ is the set of all fluents that become true after executing action a on state s

$$Adds(s, a) = Add(a) \cup \bigcup_{i, c_i \subseteq s} ca_i$$

and $Dels(s, a)$ is the set of fluents that become false after executing action a on state s

$$Dels(s, a) = Del(a) \cup \bigcup_{i, c_i \subseteq s} cd_i$$

where c_i is the condition of conditional effect $ce(a)_i$ associated with action a.

Extending Strips with Action Costs

The expressiveness of STRIPS can be easily enhanced by extending it to deal with *action costs*, so the cost of plans is not limited to the number of actions in it. One straightforward and simple way of doing so is to define costs of plans as an additive function over arbitrary positive reals:

Definition 2.8 (STRIPS with costs). *A* STRIPS *problem with costs* $P_c = \langle F, A, I, G, c\rangle$ *consists of:*

1. *The* STRIPS *problem defined by* $P = \langle F, A, I, G\rangle$*,*
2. *A cost function* $c : A \rightarrow \mathbb{R}_0^+$ *that assigns to each* STRIPS *action in the problem a non–negative cost*

Having addressed the issue of defining costs of actions – and valid plans – we can now define the notion of *optimal plan*:

Definition 2.9 (Optimal plan). *An* optimal *plan* π^* *for* STRIPS *problem* $P_c = \langle F, A, I, G, c\rangle$ *is a plan such that*

$$c(\pi^*) = \min_{\pi} c(\pi)$$

where π *is any* valid *plan for* P_c*.*

The set of best solutions, or *optimal plans*, to planning problems P is denoted $\Pi_P^*(G)$.

In this thesis, references to problems P are to STRIPS problems with costs, where the cost function is $c(a) = 1$, for all $a \in A$, unless explicitly noted.

2.3 Complexity of Strips planning

The problem of finding a minimal cost path from the single source s_0 to all $s \in S$, and therefore all $s \in S_G$ in a planning state model Π, is in **P** (Papadimitriou, 1994) and can be solved by Dijkstra's algorithm in time $O(|S|^2)$ (Cormen et al., 1989).

However, the size of state sets S in state models derived from STRIPS planning problems P are exponential on the number of fluents $O(2^{|F|})$ and typically too large even for the purposes of enumeration. While this bound might be quite loose, due to many possible states s not being reachable, attempting to solve problems P with Dijkstra [2] means dealing with not only an exponential number of states, but also with exponential length paths.

The *classical* characterization of the complexity of STRIPS planning problems is due to (Bylander, 1994), where concrete results are given for the following two decision problems:

Definition 2.10 (Plan Existence). *Given a* STRIPS *representation of a planning problem* P*,* ***PlanExt***(P) *is the following decision problem:*

[2]Or its more space efficient version, tailored to solve state models, known as Uniform Cost Search (Russell and Norvig, 2003).

`INSTANCE:` *A planning problem P.*

`QUESTION:` *Does a* valid *plan π for P exist?*

Definition 2.11 (Plan Cost). *Given a factored representation of a planning problem P and a constant value $k \in \mathbb{R}_0^+$, $\texttt{PlanCost}(P)$ is the following decision problem:*

`INSTANCE:` *A planning problem P, a constant $k \in \mathbb{R}_0^+$.*

`QUESTION:` *Does a plan π for P with cost$(\pi) \leq k$ exist?*

Both decision problems are shown to be **Pspace**–complete which is *exponentially separated* from **NP**–complete (Papadimitriou, 1994). This is broadly considered as a disappointing result by the AI research community – i.e. "planning is hopeless" – but contrasts starkly with the practice of planning, which routinely solves *concrete* planning problems shown to be in **NP** or, somewhat surprisingly, in **P** (Helmert, 2003, 2005).

Recent work (Bäckström and Jonsson, 2011) partly bridges this gap between theory and practice by noting the difference between *computational* and *expressive* power of decision problems in the same complexity class, and proposing a battery of new methods to establish a more nuanced and proper complexity characterization of planning problems.

These new methods replace those used by (Bylander, 1994) which have become the standard tool for investigating into complexity of planning problem. (Bylander, 1994) results were built on a straightforward mapping into Turing Machines such that the state transition function $f(s, a)$ was encoded *explicitly* into Turing Machine operators. This kind of representation is at odds with the standard practice of using very efficient state expansion procedures that allow to explore the state graph implicitly (Balcazar, 1996), as long as the number of successors to some particular state s is bounded by a polynomial. (Balcazar, 1996) proposes – and is followed upon by (Bäckström and Jonsson, 2011) – to encode this node expansion procedure as a *boolean circuit*, which allows for a more informative and consistent framework to study the hardness of planning problems.

2.4 Heuristics

Given the interpretation of planning problems as implicitly defined graphs whose nodes represent states and whose edges represent actions, graph search algorithms are a logical choice for solving them. Yet, as discussed in Section 2.3, general algorithms such as Dijkstra cannot effectively deal with the large state spaces usually associated with planning problems.

However, *heuristic* search approaches (Pearl, 1983) using a heuristic function computed *automatically* from planning problems encoded in STRIPS (Bonet and Geffner, 2001a) have proven to be successful in recent years, with heuristic search based planners winning several of the most recent International Planning Competitions (IPC) (Richter and Westphal, 2008; Helmert, 2004; Hoffmann and Nebel, 2001; Bonet et al., 1997).

A *heuristic* is a way of choosing between different course of action in order to achieve some goal (Pearl, 1983). Heuristics make no guarantees as to the optimality of their suggestions. Yet in combination with graph search algorithms allow to find both optimal and satisficing solutions to many problems where the search space is just too large for general search methods. Canonical examples are problems such as the 15–puzzle, in which tiles must be moved around a grid to reach a target configuration, and the route finding problem, in which a route must be found between two cities in a highway map. In many settings, heuristics take the form of *heuristic estimators* that compute, for a given state s, an estimate of the cost of reaching the goal from that state:

Definition 2.12 (Heuristic estimator)**.** *A heuristic estimator is a function* $h : S \rightarrow \mathbb{R}_0^+$ *that given a state* s *estimates the cost of reaching a goal state* $s' \in S_G$ *from* s*.*

One example of using heuristic estimators for guiding the search would be to apply first actions a in s that minimize the estimated cost $h(s')$ of the state $s' = f(s, a)$. While such a state appears to represent the quickest way of getting to the goal from the current situation, even if the estimate is known to not over–estimate the cost of reaching the goal from s, alternative actions a' must be also considered when a globally optimal solution is sought.

While its computation is impractical for problems of interest, it is useful to define the *perfect heuristic estimator* h^*. This estimator returns the *actual* cost of optimal solutions from any state s,:

Definition 2.13 (Perfect heuristic estimator)**.** *The perfect heuristic estimator* $h^*(s)$ *is the cost of an optimal (lowest–cost) solution to a problem from state* s*.*

The fact that h^* gives the optimal cost from the current state s means that there should always exist an action a such that $h^*(s) = c(a) + h^*(f(s, a))$. This in turn implies that it should be possible to go from the initial state of the problem s_0 to a goal state $s \in S_G$ by following a sequence of states for which the sum of the accumulated cost until that state and the value of h^* of the state are constant. In practice, heuristics for problems of interest are far less accurate, and require the exploration of a much larger part of the search space.

Admissible, or non–overestimating heuristic estimators, are defined with reference to the h^* estimator:

Definition 2.14 (Admissible heuristic estimator (Pearl, 1983))**.** *An admissible heuristic estimator* h *is one which satisfies* $h(s) \leq h^*(s)$ *for all states* $s \in S$*.*

An admissible heuristic estimator for the route finding problem, for example, is the Euclidean distance in the map between two cities, as this is guaranteed to be a *lower bound* on the actual minimum distance that must be driven to get from one to the other.

Admissible heuristic estimators are important for problem solving, as they can be used in combination with search algorithms that always explore first courses of action that appear to be less costly in order to obtain *optimal* solutions. In contrast, *non–admissible* heuristic estimators may lead to *suboptimal* solutions, as an overestimate of the optimal cost of a state may cause the algorithm to rule out a solution that actually has lower cost. This oversight also means that search algorithms driven by carefully designed non–admissible heuristics will find a solution much *faster*(Pearl, 1983).

Typically, both types of heuristic estimators are defined as the cost of solutions – optimal or sub–optimal – to simplified versions of the problem at hand. These simpler versions are called *relaxations* of the original problem, and result from ignoring certain of their properties or complexities (Pearl, 1983). This simplification though, comes at the expense of the precision of the estimate conveyed by the simpler problem solution, existing usually a trade–off between how close is the lower bound obtained to h^* and how expensive is to solve the *relaxation* (Helmert and Mattmüller, 2008).

There is a second class of heuristics in broad use in planning, which are not *calibrated*, that is, their suggestions are not related to the cost of solving the problem. This second class of *uncalibrated* heuristics, rather than producing estimates of costs, *rank* actions according to what degree they are deemed to be necessary in order to achieve the goal quickly. The probability of an action being the best to do in a state is usually assessed by analyzing the causal structure of the planning problem, very much as calibrated heuristics do, while being sensitive to particular structural features that are blurred by calibrated heuristics.

2.5 Heuristic search algorithms

A *heuristic search algorithm* is a procedure that uses a heuristic estimator to find a sequence of actions that reach a goal. In order to do so, an implicit directed *search graph* (Russell and Norvig, 2003) is built incrementally. Each node n corresponds with a state s plus additional information such as, the action a used to reach it and its accumulated cost. The root node n_0 corresponds to initial state s_0. This search graph allows the algorithm to keep track of the partial solutions being worked on by the search algorithm, as well as for ruling out loops.

This data structure also allows for a easy and quick retrieval of solutions found, by enumerating the actions involved in a path backwards from a goal leaf node to the root node n_0. Hence, nodes are implicitly encoding*candidate solution* paths. Children nodes, *generated* by applying action a to the state s associated to a node n, are referred to as *successor nodes.* Whenever all the successors of a given node n are inserted into the search graph, it is said that node n has been *expanded.* The set of leaf nodes in the search graph is commonly referred to as the *search frontier.*

Algorithms for solving state models $\mathcal{S}$ can be arranged along two different dimensions:

1. *Branching strategy* – The strategy followed to traverse the directed graph implicitly encoded by transition function f: breadth–first or depth–first. Both breadth and depth can be bounded, leading to *beam search* and *iterative deepening* algorithms, establishing a compromise between space, time and completeness (Korf, 1993).

2. *Information* – Whether there is available a heuristic function h which estimates the cost of the path between two states $k(s, s')$. Usually s' is fixed to be some goal state $s' \in S_G$, though this particular commitment can change as search progresses. Admissible heuristics h (Pearl, 1983) are of special importance,

since they enable algorithms which are *optimal* with respect to the amount of search done and the quality of the solution obtained after search.

The planning community has focused on *informed* or *heuristic* search algorithms, trying alternatives in directionality and branching strategy with varying degrees of success. In *optimal planning* though, the most successful systems have been for a long time those using combination of backward chaining and iterative–deepening depth–first search (Korf, 1985; Bonet and Geffner, 1999). Recently these have been superseded by forward chaining search using admissible heuristics unrelated to the h^m heuristics(Bonet and Helmert, 2010; Nissim et al., 2011).

For *satisficing planning* [3] most successful systems have been those combining non–admissible heuristics, forward chaining and best–first search. The most influential satisficing planner amongst those developed during the early 2000's was FF (Hoffmann and Nebel, 2001). FF proved to be much faster than the first satisficing planner based on heuristic search, HSP (Bonet et al., 1997). Rather than just using *best–first search* as HSP, FF followed a two–tiered strategy. First, an incomplete [4] but very fast search algorithm is invoked. This algorithm, *Enforced Hill Climbing*, combined the then novel notion of *relaxed plans* and *helpful actions* with those in *beam search*, in order to explore very quickly parts of the search space deemed by the heuristic to be the ones leading to the goal with the least cost. When this algorithm fails, then a best–first search algorithm using the same non–admissible heuristic is invoked. (Hoffmann and Nebel, 2001) delivered a powerful planner, which can be argued to have been not beaten by any other satisficing planner between 2001 and 2008.

FF lost this dominant position recently with the development of the LAMA planning system (Richter and Westphal, 2008, 2010). LAMA is an *any–time* planner which relies on a heavily modified best–first search procedure that uses dynamically several non–admissible heuristics, and integrates some of the ideas in FF to keep the search algorithm focused on relevant parts of the search space.

Best First Search

Best first search (Hart et al., 1968; Nilsson, 1980) (BFS) algorithms explore the search space by ranking all of the current leaf nodes, whose children have not yet been generated, according to some evaluation function $f(n)$. This function typically is a linear combination of the *accumulated cost* to reach the goal, denoted by $g(n)$, and the heuristic value, written as $h(n)$ (Pearl, 1983). The algorithm operates by inserting leaf nodes into the *open list*, an ordered list where nodes are stored so that those nodes with lower $f(n)$ values come first (Algorithm 2.1).

At each step of the algorithm, a node n with minimum $f(n)$ is retrieved from the open list, and its successors are generated by applying all actions $a \in app(s)$ and inserted according to their $f(n)$ value. In order to avoid exploring paths already considered, expanded nodes are placed into the *closed list*. Whenever a new node n is generated, it is checked that there is no other node n' with the same state s present

[3]Non–optimal planning where solutions are not required to be optimal but *good enough*.

[4]Incomplete search algorithms do not provide the guarantee of finding the solution to a search problem, even when such solution exists.

```
Input: A problem model of the form $\Pi = \langle S, s_0, S_G, A, app, f \rangle$
Output: A path to a goal state $s_g$
open-list ← a set of nodes;
$n_i$ ← a node representing the initial state;
Calculate $f(n_i)$ and add $n_i$ to open-list;
while open-list $\neq \emptyset$ do
    $n$ ← a node with minimal $f(n)$ in open-list;
    Remove $n$ from open-list;
    if IsGoal($n$) then
        return GeneratePath($n$);
    for $n' \in$ GenerateSuccessors($n$) do
        Calculate $f(n')$;
        if $n' \notin$ open-list, closed-list then
            Add $n'$ to open-list;
        else if previous $f(n')$ > current $f(n')$ then
            Update the path that lead to $n'$ and its $f(n')$ value;
            if $n' \in$ closed-list then
                Move $n'$ to open-list;
    Place $n$ in closed-list;
```

Figure 2.1: Best-first search algorithm.

in either the open and closed lists. If there exists such other node n', then values $f(n)$ and $f(n')$ are compared. If it is found that $f(n) < f(n')$ then n' is removed from the list in which it was found, and n is inserted into the *open* list.

A generic form of the typical evaluation function used by heuristic search algorithms is given by $f(n) = g(n) + w\dot{h}(n)$. When $w = 1$, this results in the A* (Hart et al., 1968) algorithm, which finds optimal solutions when used in combination with admissible heuristics while generating the smallest possible search tree (Pearl, 1983). When $w > 1$ the value of the heuristic for a state is given more relevance, and the search is focused on states whose heuristic value is the smallest, even if their accumulated cost is greater, resulting in the algorithm known as Weighted A*. The "greediness" of the search strategy, with respect to the "prospects" of reaching the goal, becomes extreme when $g(n)$ values are ignored, so $f(n)$ effectively becomes $h(n)$. In this last case, the algorithm receives the name of Greedy Best–First Search.

A number of refinements have been proposed for the generic best–first search strategy described above in the context of planning. We will briefly review these when discussing the Lama planner (Richter and Westphal, 2008), which has integrated the most effective of these refinements in a coherent way.

2.6 Reachability Planning Heuristics h^m

In this section we will discuss the h^m family of heuristics (Bonet and Geffner, 2001a; Haslum and Geffner, 2000; Haslum et al., 2005) which are *calibrated*, that is, the values computed correspond to estimates of costs to reach the goal. This family of heuristics are automatically derived from Strips representations of planning problems P (McDermott, 1999; Bonet and Geffner, 2001a), by combining two relaxation procedures: one that operates at a *semantic* level, ignoring part of action descriptions and a second one, which operates at a *structural* level, which ensures the relaxation to be *tractable*, since computing solutions for it becomes bounded by a *polynomial* of degree m.

Ignoring Delete Lists

The first step in the relaxation consists in obtaining a *relaxed* version of P, P^+, in the following manner:

Definition 2.15. *Given P, a* Strips *planning problem $P = \langle F, I, A, G \rangle$, a* delete–relaxation *of P, $P' = \langle F', I', A', G' \rangle$ is defined as follows:*

1. $F' = F$
2. $I' = I$
3. *Action set A' is like A, but for each $a \in A'$, $Del(a) = \emptyset$.*
4. $G' = G$

this notion of explicitly relaxing a planning problem by operating directly on the problem description, was first suggested by (Pearl, 1983).

However, P^+ alone, which is referred to with the name of *delete–free relaxation* is not useful to derive a heuristic estimate. Computing the optimal solutions of P^+ is not tractable:

Theorem 2.16. *(Bylander, 1994; Hoffmann and Nebel, 2001) Let P be a* Strips *planning problem where for all $a \in A$, $Del(a) = \emptyset$.* PlanMin *for such P belongs to* Np*–hard.*

Proof. By reduction of PlanMin to Min Set Cover. □

NP–hard is exponentially easier that **Pspace**–complete but it is still too hard.

On the other hand, computing *approximate* solutions to P^+, which can result in either lower or upper bounds on $c^*(P^+)$, the optimal cost of solutions for P^+, can be polynomial.

The simplest and most efficient of such approximations, the h_{max} and h_{add} heuristics (Bonet and Geffner, 2001a), estimate the cost of achieving goal or action precondition fluents *individually*. The heuristic estimator h is then defined to be some suitably chosen function which combines individual fluent estimates. This allows a parametric family of heuristic estimators, h^m, to be defined. m is the size of the number of goal literals being considered *simultaneously* and whose time complexity is polynomial on m. h_{max} and h_{add} correspond to the case where $m = 1$.

h^1 Heuristics

The cost to individual literals is computed by adapting dynamic–programming methods for computing multiple–source shortest paths in graphs (Ford and Fulkerson, 1962).

Definition 2.17. *Let $G^1(P) = (V, E)$ be the directed graph derived from planning problem $P = \langle F, I, A, G\rangle$ where vertex set V corresponds to fluents $p \in F$ plus a dummy source node corresponding with to state I. The set of edges E is defined as:*

$$E = \bigcup_{a \in A} \{(p, q) \mid p \in Pre(a), q \in Add(a)\} \cup \{(I, p) \mid p \in I\}$$

The cost of achieving literal p is then reflected in the length of the paths that lead to p from the graph node corresponding to I.

We will denote the cost of achieving a fluent p from state s as $h(p; s)$. These estimates can be defined recursively as

$$h(p; s) = \begin{cases} 0 & \text{if } p \in I \\ \min_{a \in O(p)}[c(a) + h(Pre(a); s)] & \text{otherwise} \end{cases} \tag{2.2}$$

where $O(p) = \{a \mid p \in Add(a)\}$, operator min over an empty set is defined to be ∞ and $h(C; s)$, with C being a set (conjunction) of fluents, is a *combination* functional parameter of the heuristic estimator, whose nature determines the admissibility of the resulting estimator.

Estimates $h(p; s)$ are obtained from a simple forward chaining procedure in which the estimates are initialized to 0 if $p \in s$ and to ∞ otherwise. Then, for every action a such that $s \models Pre(a)$, each fluent $p \in Add(a)$ is added to s and $h(p; s)$ is updated to

$$h(p; s) := \min\{h(p; s),\ c(a) + h(Pre(a); s)\} \tag{2.3}$$

These updates continue until the estimates $h(p; s)$ do not change. The procedure is polynomial in the number of literals and actions, and essentially is the same algorithm as the Bellman–Ford algorithm for finding shortest paths in graphs (Ford and Fulkerson, 1962).

As mentioned above, there are many possible ways to define $h(C; s)$ which result in different heuristics. Two such definitions have been proved to be very effective and are still in use in many classical planners: the additive heuristic h_{add} and the max heuristic h_{max} [5].

The additive heuristic is obtained by defining $h(C; s)$ as follows:

$$h_{add}(C; s) := \sum_{q \in C} h(q; s) \tag{2.4}$$

The intuition for this definition lies in assuming that literals $q \in C$ can be achieved independently by $|C|$ *sequential* plans, which can be executed in *any* order. When the assumption is true, it suffices to add up together the costs of these plans. Even in the face of the possible over counting [6] due to plans not being independent, it captures

[5]We would like to emphasize that *strictly speaking* h_{add} cannot be considered part of the h^m family of heuristic functions since it is not *admissible*. We review it nonetheless along these because its based on the same structural relaxation as the admissible h^1 (h_{max}) heuristic.

[6]The simplest example of h_{add} over counting actions is as follows. Consider action a and literals p, q in $Add(a)$. $h_{add}(\{p, q\}; s)$ would be counting action a *twice*.

roughly but cheaply, relevant structural properties of planning problems (Keyder and Geffner, 2009). It is also because of this over counting that h_{add} is not an admissible heuristic.

The admissible *max heuristic*, h_{max} or h^1, is obtained by defining $h(C; s)$ in the following way:

$$h_{max}(C; s) \overset{\text{def}}{=} \max_{q \in C} h(q; s) \tag{2.5}$$

The *max heuristic* is a lower bound on $c^*(P)$ since it is only taking into account the cost of achieving *one* of the literals $q \in C$. The rationale for h_{max} is that if all literals q can be achieved by $|C|$ *parallel* [7] plans, then suffices to take it as an estimate the highest cost.

Higher–order h^m Heuristics

Computing higher–order reachability heuristics is also equivalent to computing shortest paths on graphs $G^m(P)$ (Haslum and Geffner, 2000), where vertices, rather than being single fluents, as is the case for h^1, are sets of fluents with size *at most* m. In this $G^m(P)$ graph, there is an edge between two sets of fluents C and B whenever there exists an action a s.t.

$$B \subseteq R(C, a)$$

where $R(C, a)$ denotes the operation of regressing set C through action a, which is in turn defined as

$$R(C, a) = \begin{cases} \emptyset & C \text{ inconsistent with } a \\ (C \setminus Add(a)) \cup Pre(a) \cup Del(a) & \text{otherwise} \end{cases}$$

The set C is inconsistent with action a whenever C contains fluents $p \in Del(a)$ or there exists a fluent $q \in Add(a)$ which is not a member of A. It is easy to see that when $m = 1$ the resulting heuristic is exactly h_{max}. When $m = 2$, pairs of literals (p, q) such that $h^2(\{p, q\}) = \infty$ correspond to pairs of fluents p, q which cannot be simultaneously true in *any* valid state. The h^m heuristics with $m \geq 2$ elegantly account for the notion of *mutex sets* first proposed and used by GraphPlan (Blum and Furst, 1995). While h^1 heuristics h_{max} and h_{add} completely disregard delete lists, higher order h^m heuristics implicitly do in a limited way.

For the h^2 heuristic (Haslum and Geffner, 2000), where $m = 2$, the cost of evaluating h on a search state becomes $O(|F|^2)$, making them considerably more expensive than h^1 heuristics. The planning community has yet to come up with an effective h^2–based heuristic for satisficing planning, where the heuristic estimator is not required to be admissible. This has in effect resulted in the higher order h^m heuristics being banished altogether to optimal planning, and these have received little attention until very recently (Haslum, 2009; Haslum et al., 2011).

2.7 Relaxed Plan Heuristics

On top of the basic h^m heuristics, more informed heuristic estimators can be defined relying on the observation that the computation of $h(C; s)$ for sets of fluents $p \in C$

[7]Plans whose actions a whose effects do not affect $Pre(b)$ or $Add(b)$ of actions b in some other plan.

induces a *critical* path of actions branching out from fluents in that set (Haslum et al., 2005). This was first noted by (Hoffmann and Nebel, 2001) who introduced the notion of *best supporter* of a fluent p. Best supporters are the actions a with lowest $h(Pre(a))$ and $p \in Add(a)$, and exploited it by introducing a backwards chaining algorithm which computed the *relaxed plan*, π^+, a valid plan in P^+, for fluent set C. The relaxed plan heuristic is then defined as the sum of the costs of the actions in plan π^+

$$h^+(s) = \sum_{a \in \pi^+} c(a) \tag{2.6}$$

where π^+ is the relaxed plan extracted from state s.

This idea was later extended by (Keyder and Geffner, 2008), who generalized Hoffmann and Nebel's (Hoffmann and Nebel, 2001) definition to account for any heuristic using the same kind of structural relaxation as the one used by h^1 heuristics.

Given a best supporter function, the algorithm shown in Figure 2.2 extracts a plan with at most a single instance of each operator.

```
Input: A planning problem $\Pi = \langle F, I, O, G \rangle$
Input: A best supporter function BestSupporter : $F \mapsto O$
Output: A relaxed plan $\pi$ or DEAD-END
$\pi \leftarrow \emptyset$;
$supported = \emptyset$;
$to\text{-}support \leftarrow G$;
while $to\text{-}support \neq \emptyset$ do
    Choose $p \in to\text{-}support$;
    $to\text{-}support \leftarrow to\text{-}support \setminus \{p\}$;
    if $p \notin I$ then
        if BestSupporter($p$) = $undefined$ then
            return DEAD-END;
        $\pi \leftarrow \pi \cup \{$BestSupporter($p$)$\}$ ;
        $supported \leftarrow supported \cup \{p\}$;
        $to\text{-}support \leftarrow to\text{-}support \cup (Pre($BestSupporter($p$)$) \setminus supported)$;
return $\pi$
```

Figure 2.2: Relaxed plan extraction algorithm.

As long as the plan represented by the best supporter function is well-founded (*i.e.* does not contain cycles), this algorithm will terminate. Several ways of selecting best supporters have been proposed that minimize different measures on the paths to each fluent.

a_p^{add} denotes the best supporter chosen for fluent p by h^{add}, the h^1 additive heuristic and $h^{\text{add}}(a_p^{add}; s)$ denotes the cost of applying this action. These are given by the following:

$$\begin{aligned} a_p^{add} &\overset{\text{def}}{=} \text{argmin}_{a \in O(p)} h^{\text{add}}(a;s) \qquad (2.7)\\ h^{\text{add}}(a;s) &\overset{\text{def}}{=} \mathit{cost}(a) + h^{\text{add}}(\mathit{Pre}(a);s)\\ h^{\text{add}}(Q;s) &\overset{\text{def}}{=} \sum_{q \in Q} h^{\text{add}}(q;s) \end{aligned}$$

For the admissible h^1–based max heuristic, the best supporter and estimated cost of applying an action are similarly defined:

$$\begin{aligned} a_p^{max} &= \text{argmin}_{a \in O(p)} h^{\max}(a;s) \qquad (2.8)\\ h^{\max}(a;s) &\overset{\text{def}}{=} \mathit{cost}(a) + h^{\max}(\mathit{Pre}(a);s)\\ h^{\max}(Q;s) &\overset{\text{def}}{=} \max_{q \in Q} h^{\max}(q;s) \end{aligned}$$

Relaxed plan heuristics are not admissible, so they can only be used for satisficing planning. While being relatively inexpensive to compute – once the *base* h^1 heuristic is computed – they provide accurate estimates in a surprisingly high number of planning problems, and have become the key component of state–of–the–art planners. Besides that, relaxed plan heuristics also provide us with insight into the *causal structure* of planning problems, which can be exploited with notions such as *helpful actions* (Hoffmann and Nebel, 2001; Helmert, 2006).

2.8 Heuristics Based on Planning Landmarks

Landmarks (Hoffmann et al., 2004; Richter et al., 2008; Keyder et al.) in a planning problem are necessary features of solutions to planning problems. Fluent landmarks are formulas over the set of fluents of the problem that must be satisfied by some state that occurs during the execution of any valid plan:

Definition 2.18 (Fluent landmarks)**.** *A fluent landmark L is a formula over the set of fluents F of a planning problem, such that any valid plan $\pi = \langle a_1, \ldots, a_n \rangle$ has a prefix $\pi' = \langle a_1, \ldots, a_i \rangle$, possibly of length 0, whose application in the initial state results in a state in which L is true, i.e. $s_0[\pi'] \models L$.*

These formulas are usually encoded in a very loose language, whose semantics are similar to the Linear Temporal Logic unary operator $\diamond p$, stating p to be *necessarily* true at some point during the execution of any valid plan. In practice, landmarks are expressed as either single literals or disjunctions of literals. Of special interest are binary ordering relations between fluent landmarks, since they are essential to figure out how to decompose planning problems into simpler ones.

It can be seen by choosing the empty prefix $|\pi'| = 0$ that all formulas true in the initial state of the problem are landmarks. Similarly, choosing the prefix to be the entire plan π implies that all formulas entailed by the goal are landmarks for the problem as well.

The notion of fluent landmarks can be readily extended into actions:

Definition 2.19 (Action landmarks)**.** *An operator landmark L is a formula over the set of actions A of a planning problem, such that when any valid plan π is interpreted as a truth assignment to the set of actions in the problem, with those operators appearing in π having the value* true *and those not appearing in π having the value false, L is satisfied.*

The problem of deciding whether *any* given formula, over fluents or actions, is a landmark for a planning problem is the `Landmark` problem:

Definition 2.20 (Landmark)**.** *Given a planning problem P, and a formula L over the set of fluents F or the set of operators A, `Landmark`(Π, L) is the following decision problem:*

`INSTANCE:` *A planning problem P and a landmark formula L.*

`QUESTION:` *Is L a landmark for P?*

The `Landmark` problem is known to be **Pspace**–complete (Hoffmann et al., 2004) even when formulas are restricted to size 1, e.g. single fluent and single action landmarks. Approaches to landmark finding therefore focus on finding landmarks for the delete relaxation P^+ of planning problems, in which setting the problem of deciding `Landmark`(P^+, L) when L is restricted to single fluent and operator landmarks can be shown to be in P (Hoffmann et al., 2004).

Landmark–based heuristics (Richter et al., 2008) exploit this result to obtain partially–ordered sets of fluent landmarks. This list is then checked against the plan prefix leading to state s during search to obtain a numeric estimate of how many landmarks still need to be satisfied. This is an uncalibrated heuristic, since its values do not correspond to costs to reach the goal, but rather refer to a general notion of *progress* towards a particular goal.

Landmark heuristics have been proven experimentally to complement nicely the shortcomings inherent to h^m heuristics, especially the heuristics derived from graphs similar to those h^1 operates on, which tend to exhibit *plateaus* when the goal involves achieving independently fluents which are achieved by diverging plans. Increasing m makes this problem to go away to some extent, but computing these stronger heuristics conveys a severe performance hit during search.

2.9 State–of–the–art Planners

The LAMA planner (Richter and Westphal, 2008) has been the undisputed winner in the last two International Planning Competitions satisficing tracks, held in 2008 [8] and 2011[9]. We will devote this section to describe how all of the ideas discussed so far coalesce into a robust and scalable satisficing planner.

[8]Detailed 2008 competition results can be found here: `http://ipc.informatik.uni-freiburg.de/Results`

[9]For the 2011 competition results check: `http://www.plg.inf.uc3m.es/ipc2011-deterministic/Results`

LAMA can be broken down into three separate components and we will focus on discussing the last one, which together with the already discussed landmark heuristics, constitute the most relevant features to explain LAMA outstanding performance:

1. PDDL planning problem description parsing and compilation into SAS$^+$ (Bäckström and Nebel, 1995), a language more compact than STRIPS but otherwise semantically equivalent.
2. Domain–independent heuristics portfolio, with efficient implementations of the relaxed plan heuristic discussed in Section 2.7 based on the h_{add} heuristic, along with a similarly defined relaxed plan heuristic based in the notion of landmarks discussed in Section 2.8.
3. A search engine implementing a sophisticated variant of Best–First Search, that accommodates any–time behavior and sophisticated node expansion and evaluation strategies which effectively exploit and enhance the information conveyed by the heuristics in the portfolio.

Anytime algorithms are those that return a sequence of approximations to the optimal solution such that each approximation is not worse than the previous one, i.e. the algorithm can be stopped at *any time*. In order to do so, the search becomes less greedy with respect to heuristic estimates as new candidate solutions are proposed. The evaluation function $f(n)$ used by LAMA during search is *dynamic*. It starts with $f(n)$ defined to be $f(n) = h(n)$ while looking for the first solution, and switches to $f(n) = g(n) + (W \cdot k)h(n)$ in each successive iteration, where k is a constant and $W\dot{k}$ is defined to be ≥ 1. Besides that, the cost of the solution found in the i-th search, $c(\pi_i)$, is used to prune nodes n with $g(n) \geq c(\pi_i)$ expanded during the $i + 1$-th search. This effectively turns LAMA into an explicit branch & bound search algorithm, where the probability of finding an optimal solution grows with each search performed. Indeed, whenever LAMA reports no solution in any search other than the first one, the previous solution reported is guaranteed to be an *optimal* one (Richter and Westphal, 2010). This strategy aims at minimizing the amount of search done and can indeed result in LAMA finding an optimal solution faster than optimal planners in some planning problems.

The notion of *helpful action* (Hoffmann and Nebel, 2001) is used in a novel way in the search. Rather than having one single open list as in the classical definition of Best First Search, LAMA keeps as many as $2n$ open lists, where n is the number of heuristics in the portfolio. For each of the heuristics, two lists are kept, one to store nodes generated when applying *helpful actions*, and another for nodes generated with actions not in the helpful action set, both sharing the same definition for $f(n)$. When a node is to be expanded, both lists are alternated with a frequency which depends on whether successors for the node selected for expansion decreased the minimum $f(n)$ value for *both* sets of lists (Richter and Westphal, 2010).

Computing heuristic estimators' values is typically the bottleneck in heuristic search planners, and computing resources are wasted when a large proportion of the nodes generated are never going to be considered for expansion. *Delayed evaluation* (Helmert, 2006; Richter and Westphal, 2010) seeks to minimize the impact of this issue, by not evaluating the heuristic estimator for node n until it is expanded, using node n's parent heuristic value in $f(n)$ in the meantime. While this might lead to poorer ordering of individual nodes, this effects tends to be offseted by the ability to generate a larger part of the search space in less time.

2.10 Example: RoboSchool in Strips

Modeling a planning task with the STRIPS language can be broken down into several steps defining the following:

1. The objects relevant to the task, their attributes and relations with other objects, which describe the state of the world,
2. The initial state and goal formula,
3. The actions in terms of their arguments, preconditions and effects.

We will discuss each step in order.

Describing World States: Objects & Predicates

For this particular task, it is straightforward to identify the objects we are interested in tracking. Table 2.1 shows them grouped into sets, each set corresponding to a PDDL *type* (McDermott, 2000).

Type	Objects
pieces	$p_1, p_2, p_3, \ldots, p_7$
locations	$A0, \ldots, A4, \ldots, E0, \ldots, E4$
bins	b_1, b_2, b_3
shapes	$box, sphere, cone$
colors	$red, blue, green$

Table 2.1: Objects in ROBOSCHOOL task depicted on Figure 1.1.

Elements in each set, and in the implicitly defined superset including all objects, are denoted with *variable* symbols. We will use o to refer to *any* object, p for pieces, l for locations, b for bins, s for shapes and c for colors. ROBOSCHOOL predicates and their meaning are shown in Table 2.2.

Predicate	Meaning
trainerAt(l)	Trainer is at location l.
pieceAt(p, l)	Piece p is at location l.
binAt(b, l)	Bin b is at location l.
carries(p)	Trainer carries piece p.
inside(p, b)	Piece p is inside bin b.
connected(l_i, l_j)	The trainer can navigate from l_i to l_j.
adjacent(l_i, l_j)	Locations l_i and l_j are adjacent.

Table 2.2: ROBOSCHOOL predicates used to describe world states.

Examples of fluents describing the situation depicted in Figure 1.1 would be, pieceAt$(p_2, A1)$ accounting for the green box at cell $A1$, connected$(C0, C1)$ that would account for the ability to freely move between $C0$ and $C1$ or adjacent$(B4, C4)$ to account for the fact that cell $C4$, though inaccessible, is adjacent to the accessible cell $B4$.

Initial and Goal States

The initial state I of our planning problem corresponds with the transcription, in terms of fluents, of the situation depicted on Figure 1.1. A fragment of the initial state representation, describing *piece* object locations would be:

$$I = \{\ldots, \text{pieceAt}(p_2, A1), \text{pieceAt}(p_3, A4), \text{pieceAt}(p_1, B1) \ldots\}$$

Goals in RoboSchool are also encoded as conjunctions (sets) of fluents. Pddl allows to specify these as universally or existentially quantified formulas (McDermott, 2000), and state–of–the–art planners can rewrite them into plain Strips with ease (Gazen and Knoblock, 1997) as follows:

1. "Store all triangles in b_1" becomes
$$G_1 = \{inside(p_3, b_1), inside(p_4, b_1)\}$$
2. "Store all spheres in b_2" becomes
$$G_2 = \{inside(p_1, b_2),\ inside(p_7, b_2)\}$$
3. "Store all cubes in b_3" becomes
$$G_3 = \{inside(p_5, b_3),\ inside(p_6, b_3)\}$$
4. "Store red objects in b_2" becomes
$$G_4 = \{inside(p_1, b_2),\ inside(p_3, b_2),\ inside(p_6, b_2)\}$$
5. "Store green objects in b_3" is
$$G_5 = \{inside(p_2, b_3),\ inside(p_7, b_3)\}$$
6. And "Store blue objects in b_1" becomes
$$G_6 = \{inside(p_4, b_1),\ inside(p_5, b_1)\}$$

Describing Behavior: Actions

The usual practice when modeling planning problems is to use Pddl action schemas (McDermott, 2000), which are *lifted* representations of Strips ground actions. Action schemas consist of the following elements:

1. Action *name*,
2. An *argument* list, a list of typed variable names denoting objects,

3. A *precondition*, a first–order formula defined over the variables in the argument list and the predicates previously defined,
4. An *effect* formula, also defined over argument variables and predicates.

Very much like goals, PDDL allows preconditions and effects to be defined with a very expressive language encompassing standard logical operators – $\wedge$, $\vee$, $\supset$, $\neg$ – as well as quantifiers. Most planners support the full set of operators, grounding action schemas and rewriting the formulas into DNF normal form as described by (Gazen and Knoblock, 1997).

Action	Preconditions	Adds	Deletes
walk(l_1, l_2)	trainerAt(l_1) connected(l_1, l_2)	trainerAt(l_2)	trainerAt(l_1)
pickUp(p, l)	trainerAt(l) pieceAt(p, l)	carries(p)	pieceAt(p, l)
drop(p, l, b)	trainerAt(l) carries(p) binAt(b, l)	inside(p, b)	carries(p)
throw(p, l_1, l_2, b)	trainerAt(l_1) carries(p) binAt(b, l_2) adjacent(l_1, l_2)	inside(p, b)	carries(p)

Table 2.3: ROBOSCHOOL STRIPS action set A. p symbols denote pieces, b symbols denote bins and l symbols are cells.

Table 2.3 shows the description of ROBOSCHOOL actions already compiled into STRIPS. Action costs for *walk* actions going up, down, left and right, as well as, *pickUp* and *drop* are 1. *walk* actions corresponding to moves along diagonals cost $\sqrt{2}$. *throw* actions have a cost of 2.

Planning Problems & Plans

Solving a ROBOSCHOOL plan recognition problem actually involves considering five different planning problems, one for each of the hypothetic goals $G_1, \ldots, G_5$, $P[G_1]$, ..., $P[G_5]$ and each of them with their corresponding set of valid plans $\Pi_P(G_1)$, ..., $\Pi_P(G_5)$. An optimal plan π^* for planning problem $P[G_1]$ would be:

$$\begin{aligned}\pi &= (walk(A0, B0),\, pickUp(p_4, B0),\\ &\quad walk(B0, C1),\, throw(p_4, C1, D2, b_1))\end{aligned}$$

with a total cost of $1+1+\sqrt{2}+2 = 4+\sqrt{2}$. Obtaining the plans for all of the $P[G_i]$ problems above takes the LAMA planner less than 2 seconds.

2.11 Summary

We have offered an in-depth review of several key concepts such as optimal plans and illustrated the features offered by planning factored representations that can be considered as the foundations for the formulation of plan recognition given in Chapter 4. We have also reviewed the search algorithms and heuristics used by state–of–the–art classical planners that allow these to scale up notably well, a fact we exploit to solve plan recognition problems efficiently.

Chapter 3

Goal MDPs and POMDPs Planning

In this chapter we will present models that account for non–deterministic actions effects and partial feedback on the effect of actions on states. We will start by discussing Markov Decision Processes (MDPs), planning models with probabilistic action effects and full state observability and review the dynamic programming algorithms that solve them optimally. Special attention will be given to the Labeled Real–Time Dynamic Programming (Bonet and Geffner, 2003) algorithm, implemented in the GPT (Bonet and Geffner, 2001b) planner that we use in the experiments we discuss in Chapter 5. After that we will introduce a more expressive model that supports partially observable world states, Partially–Observable Markov Decision Processes (POMDPs), and show how POMDPs can be solved by mapping them onto a MDP defined over *belief states.* We end with an overview of the language used by GPT to represent MDPs and POMDPs in a compact way.

3.1 Goal Markov Decision Processes

Shortest-path or *goal* MDPs provide a generalization of the state models traditionally used in heuristic search and planning in AI, accommodating stochastic actions and full state observability. In this thesis we used the definition given by (Bertsekas, 1995) extended to account for action costs.

Definition 3.1 (Shortest–path MDP)**.** *A* shortest–path *or* goal MDP $M = \langle S, s_0, S_G, A, P_a, c\rangle$ *is a six–element tuple consisting of:*

- *A* discrete, finite and non–empty *state space S,*
- *A known* initial *state* $s_0 \in S$
- *A non-empty set of* goal states $S_G \subseteq S$*,*
- *A set of* actions A*, and* applicability *function* $A(s)$*,*
- *A* probability distribution $P_a(s'|s)$ *for* $a \in A$*,* $s, s' \in S$*, and*
- *A positive* cost function $c(a, s) > 0$ *for* $a \in A$ *and* $s \in S \setminus S_G$*.*

We note that MDPs M and classical planning models Π differ only in that the transition function $f(a, s)$ is replaced by $|A|$ probability distributions $P_a(s'|s)$. States s' resulting from doing action a in state s can no longer be predicted with full certainty, but are assumed to be *fully observable*, thus providing *feedback* for deciding what action a' to execute in s'.

Due to the presence of feedback, the solution of an MDP is no longer an action sequence but a *policy*:

Definition 3.2. *A* policy *is a function $\pi : S \to A$ mapping states s into actions a such that $\pi(s) \in A(s)$.*

Policies implicitly assign probabilities to state trajectories $t = (s_0, \ldots, s_n)$

$$P(t|\pi, s_0) = \prod_{i=0}^{n-1} P_{\pi(s_i)}(s_{i+1}|s_i) \tag{3.1}$$

Goal states $s \in S_G$ are assumed to be absorbing and cost-free; meaning $P_a(s|s) = 1$ and $c(a, t) = 0$ for all $a \in A$. We can define then the *expected cost* of a policy starting its execution as:

Definition 3.3 (Expected cost of π)**.** *The* expected cost *of executing a policy π starting in state s is*

$$V^\pi(s) = \sum_{t \in T[s]} P(t|\pi)c(t)$$

where $T[s]$ is the set of state trajectories $t = (s_0, s_1, \ldots)$ with $s_0 = s$ and $c(t)$ is

$$c(t) = \sum_{i=0}^{\infty} c(\pi(s_i), s_i)$$

Policies π with lower V^π are preferred, so policies guaranteeing the lowest V^π are *optimal solutions* of Goal MDP M:

Definition 3.4 (Optimal policy π^*)**.** *An* optimal policy *π^* is a policy with* minimum *expected cost $V^\pi(s)$ over* all *states $s \in S$.*

Optimal policies existence is guaranteed when the conditions in the following Theorem are met by M:

Theorem 3.5 ((Bertsekas, 1995))**.** *An optimal policy π^* is guaranteed to exist for Stochastic Shortest–Path* MDP *M if there exists, for every pair of states $s \in S$ and $s' \in S_G$, a finite sequence of actions $\alpha = (a_1, \ldots, a_n)$ such that $P_\alpha(s'|s) > 0$.*

However, such optimal solutions, very much like optimal solutions to the classical planning model, do not need to be *unique.*

Stochastic shortest-path and Goal MDPs appear to be less expressive than *discounted reward* MDPs (Puterman, 1994; Bertsekas, 1995), where there is no goal, rewards can be positive, negative, or zero, and a parameter γ, $0 < \gamma < 1$, is used to discount future rewards. Yet, the opposite is true: discounted reward MDPs can be transformed into equivalent Goal MDPs, but the opposite transformation is not possible (Bertsekas, 1995). The same is true for discounted reward Pomdps and Goal Pomdps (Bonet and Geffner, 2009).

We are interested in computing optimal, or near–optimal, policies. Since we are assuming the initial state s_0 to be known, it is sufficient to compute a *partial policy that is closed* relative to s_0. Such a partial policy only prescribes the action $\pi(s)$ to be taken over a subset $S_\pi \subseteq S$ of states in M:

Definition 3.6 (Partial policy)**.** *A policy π is a* partial *when π is defined as*

$$\pi : S_\pi \rightarrow A$$

where $S_\pi \subseteq S$.

such policies define a set of reachable states $S(\pi, s)$:

Definition 3.7 (Reachable states of π)**.** *The set of* reachable states *of a policy π from a state s is the set:*

$$S(\pi, s) = \{s' \,|\, P(s'|s, \pi) > 0\}$$

where $P(s'|s, \pi)$ is the probability of reaching s' from state s when following π. Reachable states for an optimal policy π^* are the *relevant states* of M.

Partial policies of interest are those which are *closed*, that is, are guaranteed to take actions which lead to states the policy is defined for:

Definition 3.8 (Closed policy)**.** *A partial policy π is closed with respect to state s if and only if $S(\pi, s) \subseteq S_\pi$.*

3.2 Solving MDPs by Dynamic Programming

Any heuristic function h defines a *greedy policy* π_h:

$$\pi_h(s) = \operatorname*{argmin}_{s \in A(s)} c(a, s) + \sum_{s' \in S} P_a(s'|s) h(s') \tag{3.2}$$

where the expected cost from the resulting states s' is assumed to be given by $h(s')$. We call $\pi_h(s)$ the *greedy action* in s for the value function h. If we denote the optimal expected cost from a state s to one of the goal states S_G by $V^*(s)$, it is well known that the greedy policy π_h is *optimal* when h is the *optimal cost function* $V^*(s)$.

We note that the greedy policy π_h is *not* unique, due to the presence of ties in Equation 3.2. We will assume through the thesis that these ties are broken systematically using a static ordering on actions a. As a result, every value function V defines a *unique* greedy policy π_V, and the *optimal* cost function V^* defines a unique optimal policy π_{V^*}.

Computing optimal cost functions V^* amounts to solving a system of $|S|$ fixed point equations (Bellman, 1957):

$$V(s) = \min_{a \in A} \left\{ c(a, s) + \sum_{s' \in S} P_a(s'|s) V(s') \right\} \tag{3.3}$$

for all $s \in S \setminus S_G$. This system of equations can be solved by the standard dynamic programming *Value Iteration* (VI) method, that solves Equation 3.3 by plugging an initial arbitrary guess in the right–hand side of Equation 3.3 and obtaining a new guess on the the left–hand side. In the form of value iteration known as *asynchronous* Value Iteration (Bertsekas, 1995), this operation can be expressed as

$$V(s) := \min_{a \in A} \left\{ c(a,s) + \sum_{s' \in S} P_a(s'|s) V(s') \right\} \tag{3.4}$$

where V is a vector of size $|S|$, which is set to $V(s) = 0$ for $s \in S_G$ and arbitrarily for $s \in S \setminus S_G$, and where we have replaced the equality in Equation 3.3 by an assignment. The use of Equation 3.4 for updating a state value in V receives the name of state update or simply *update*.

In standard, or *synchronous*, Value Iteration, all states are updated in *parallel* while in asynchronous Value Iteration, only a selected subset of states is selected for update. In either case it is guaranteed that V will eventually converge to the optimal value function if all states are updated infinitely often (Bertsekas, 1995). However, for Goal Mdps M this is true only when the assumption that *states S_G are reachable with positive probability from every state s*, holds (Bertsekas, 1995).

In practice, asynchronous Value Iteration is stopped when the Bellman error or *residual* over all states s is sufficiently small:

Definition 3.9 (Bellman error (residual))**.** *The Bellman error or* residual *is defined to be the absolute value of the difference between the left and right hand sides of the Bellman equation:*

$$R(s) \overset{def}{=} \left| V(s) - \min_{a \in A} \left\{ c(a,s) + \sum_{s' \in S} P_a(s'|s) V(s') \right\} \right|$$

In the discounted Mdp formulation, a bound on the policy *loss* – the difference the expected cost of the policy π and the expected cost of the optimal policy π^* – can be obtained as a simple expression of the discount factor γ and the maximum residual $V^{\pi}(s) - V^{\pi^*}(s)$. In Goal Mdps M there is no similar closed–form bound, although it can be computed at some expense (Bertsekas, 1995; Hansen and Zilberstein, 2001). Thus, one can execute Value Iteration until the maximum value for $R(s)$ over all states s becomes smaller than some bound ϵ.

For these reasons, we will refer in this thesis to the task of computing value functions $V^*(s)$ as that of computing $V(s)$ with residuals $R(s)$ smaller than or equal to some given positive parameter ϵ. Since implicit in the definition of M is that the initial state s_0 is known, it will be enough to consider that Value Iteration has converged when residuals $R(s)$ over states $s \in S(\pi_V, s_0)$ are smaller than ϵ.

3.3 DP Algorithms for Goal Mdps

This section reviews Labelled Real–Time Dynamic Programming (Lrtdp), the algorithm at the heart of Gpt, following closely the account in (Bonet and Geffner, 2003). Lrtdp is an extension of Rtdp (Barto et al., 1995) an implementation of the concept of asynchronous Value Iteration that computes V^* in an on–line, *anytime* manner. We will first give a brief overview of RTDP and its theoretical and practical convergence properties. After that LRTDP will be described, emphasizing how it exploits the structure of the state space S to improve significantly over RTDP convergence times. We will finally describe the domain–independent heuristic used by GPT to initialize V, that has a substantial impact on LRTDP efficiency.

Real–time Dynamic Programming

RTDP is a simple dynamic programming algorithm that involves a sequence of trials or runs, each starting in the initial state s_0 and ending in a goal state. Figure 3.1 shows a pseudo–code description of one RTDP trial.

```
Input: A state $s$
Input: Value function $V$
while $s \notin S_G$ do
    $a^* \leftarrow \text{argmin}_{a \in A(s)}\, c(a,s) + \sum_{s'} P_a(s'|s)V(s')$ ;
    $V(s) \leftarrow c(a^*,s) + \sum_{s'} P_{a^*}(s'|s)V(s')$;
    Pick $s'$ with probability $P_{a^*}(s'|s)$;
    $s \leftarrow s'$;
```

Figure 3.1: RTDP trials

Each RTDP trial is the result of simulating the greedy policy π_V while updating the values $V(s)$ using Equation 3.4 over the states s that are visited. Thus, RTDP is an asynchronous value iteration algorithm in which a single state is selected for update at each iteration. This state corresponds to the state visited by the simulation, that samples a successor state s' from the distribution of possible successors $P_a(s'|s)$. This simple scheme turns out to be surprisingly strong when compared with general asynchronous value iteration. Some of their properties are discussed next.

Theorem 3.10. *(Bertsekas and Tsitsiklis, 1996) If states S_G of M are reachable with positive probability from every state $s \in S$, then* RTDP *must reach a goal state in a finite number of steps.*

Theorem 3.11. *(Barto et al., 1995) If states S_G of M are reachable with positive probability from every state $s \in S$ and the initial value function V is* admissible, *i.e. $V(s) < V^*(s)$ for all states $s \in S$, repeated* RTDP *trials will eventually yield optimal values $V(s) = V^*(s)$ over all* relevant *states.*

Unlike asynchronous value iteration, RTDP does *not* require states to be updated infinitely often. Indeed, some states may not be updated at all. On the other hand, the proof of optimality in (Barto et al., 1995) lies in the fact that *relevant* states will be so updated.

For an efficient implementation of RTDP, a hash table T is needed for storing the updated values of the value function V. Initially the values of this function are given by an heuristic function h and the table is empty. Then, every time a value $V(s)$ for a state s is retrieved that was not previously stored in the table, a new entry for it is created. For states s in the table T, $V(s) = T(s)$ and for those not in the table, $V(s) = h(s)$.

The convergence of Rtdp is asymptotic, and indeed, the number of trials before convergence is not bounded. For instance, low probability transitions are taken eventually by some trial but there is always a positive probability that they will not be taken after any arbitrarily large number of trials. Thus for practical reasons, like for value iteration, the *termination* of Rtdp is defined in terms of a bound $\epsilon > 0$ on the residuals $R(s)$ (Bonet and Geffner, 2003).

Labeled Real–time Dynamic Programming

Rtdp behavior can be divided into two different phases (Bonet and Geffner, 2003): a *improvement* phase, where the expected costs V^{π} conveyed by the value function $V(s)$ under approximation improve rapidly, and a *convergence* phase, which involves a much larger number of trials as V converges to V^{*}. Both phases are a consequence of the *exploration* strategy inherent to Rtdp, *greedy* simulation, which privileges heavily the most likely trajectories t resulting from the greedy policy π_V. These are indeed the most relevant states given the current value function V, and it is the reason why updating them produces a substantial impact on V^{π}. On the other hand, states lying along less likely paths are also needed for convergence, but these appear with much lower frequency during trials.

Lrtdp (Bonet and Geffner, 2003) improves on Rtdp convergence by keeping track, in a systematic way, of states s for which the convergence condition $R(s) \leq \epsilon$ already holds and avoid visiting those states – and its descendants – again, resulting in dramatically shorter trials during the *convergence* phase. In order to accomplish this, a labeling procedure is introduced that systematically traverses the *greedy* graph:

Definition 3.12 (Greedy graph). *The* greedy graph $G_V = (N, E)$ *is the graph induced by the greedy policy* π_V *over the set of states* S_{π_V}*, where vertices correspond with reachable states* $N = S_{\pi_V}$*, and edges are given by:*

$$E = \{(s, s') \mid P_{\pi_V(s)}(s'|s) > 0,\ s, s' \in N\}$$

The labeling procedure keeps two Lifo data structures, named Open and Closed, respectively used to keep track of states whose residual $R(s)$ needs to be checked and to avoid cycles and duplicate work. At each step, a state s is extracted from Open, inserted into Closed and its residual $R(s)$ is checked against ϵ. If $R(s) \leq \epsilon$ the action π_V is used to generate successor states s' of s. Each successor state s' is then inserted into Open if it has not been already visited or it is still labeled as Not Solved. If $R(s) > \epsilon$, no states are added to Open. When there are no more states in Open, and no state s was found with $R(s) > \epsilon$, then all states in Closed are labeled as Solved. Otherwise, each state in Closed is updated as per Equation 3.4 in the reverse order that they were visited.

Time and space complexity of Lrtdp is thus $O(|S|)$ in the worst–case, while a tighter bound is given by $O(|S(\pi_V, s)|)$, which might be much smaller in certain domains and depending on the quality of the heuristic used to initialize V values (Bonet and Geffner, 2003).

This labeling procedure has some interesting properties, given that V is a *monotonic* value function:

Definition 3.13. *(Bonet and Geffner, 2003) A value function V is* monotonic *iff*

$$V(s) \leq \min_{a \in A(s)} c(a,s) + \sum_{s' \in S} P_a(s'|s)V(s')$$

for every $s \in S$.

the updates in Equation 3.4 will always have a *non–decreasing* effect on V. As a result, the following property can be established:

Theorem 3.14. *(Bonet and Geffner, 2003) Assume V to be an admissible and monotonic value function. Then, a call to the labeling procedure above either labels a state s as* SOLVED*, or increases the value of some state by more than ϵ while decreasing the value of none.*

A consequence of the above theorem is that the labelling procedure *alone* can be used to solve Goal MDPs M (Bonet and Geffner, 2003). However, in practice, there are two problems with using it as a Goal MDP solver. First, it will label other states as SOLVED before labeling so s_0, unless G_V is a strongly connected graph. Second, states that are close to the goal will normally converge faster than states that are farther.

LRTDP (Bonet and Geffner, 2003) algorithm results from combining the labeling procedure discussed above with RTDP. LRTDP trials are very much like RTDP trials except that these terminate whenever a state s labeled as SOLVED is found, and the labeling procedure is invoked in reverse order, from the last visited unsolved state back to s_0. LRTDP not only inherits RTDP properties of convergence in finite time, but also guarantees that it will converge in a bounded number of trials:

Theorem 3.15. *(Bonet and Geffner, 2003) Provided that at least one goal state $s' \in S_G$ is reachable from every state s in M, and the initial value function V is* admissible *and* monotonic*, then* LRTDP *solves M in a number of trials bounded by*

$$\frac{1}{\epsilon} \sum_{s \in S} V^*(s) - h(s)$$

Domain–independent Heuristics for Real–Time Dynamic Programming

Both RTDP and LRTDP greatly benefit from having V value function initialized with an *admissible* heuristic. GPT used the *domain–independent* heuristic described in (Bonet and Geffner, 2003), which is based on solving Bellman equations on a relaxation M' of the Goal MDPM. This relaxation consists of doing away with probabilistic results of actions and assuming that the most beneficial outcome of an action will take place.

Definition 3.16 (V_{min} Heuristic (Bonet and Geffner, 2003))**.** *The optimistic determinization heuristic h_{min} is defined in terms of the value function:*

$$V_{min}(s) \overset{def}{=} \min_{a \in A(s)} c(a,s) + \min_{s' : P_a(s'|s) > 0} V_{min}(s')$$

for non–goal states $s \in S$, being $V_{min}(s) = 0$ for goal states $s \in S_G$.

This heuristic can be easily shown to be both admissible and monotonic (Bonet and Geffner, 2003). In order to compute it GPT uses a variant of the LRTA* (Korf, 1990) algorithm, modified by adding the labeling procedure used by LRTDP. Interestingly, computing h_{min} accounts for a substantial part of the time required by LRTDP to solve MDP models (Bonet and Geffner, 2003).

3.4 Goal POMDPs

Goal POMDPs (Partially Observable Goal MDP) generalize Goal MDPs by modeling agents that have incomplete state information (Sondik, 1971; Monahan, 1983; Kaelbling et al., 1999) in the form of a prior belief b_0 that expresses a probability distribution over S, and a *sensor model* $Q_a(o|s)$ that describes the probability of observing $o \in O$ upon entering state s after doing a.

Definition 3.17 (Goal POMDP (Bonet and Geffner, 2009))**.** *A Goal* POMDP M_{Obs} *is a tuple* $M_{Obs} = \langle S, b_0, S_G, A, P_a, c, Obs, Q_a \rangle$ *where each element is defined as follows:*

- *A non–empty, discrete and finite* state space S*,*
- *An* initial belief *state* b_0*, probability distribution* $P(s)$ *over all* $s \in S$*,*
- *A non–empty set of* goal *states* $S_G \subseteq S$*,*
- *A set of* actions A*, and* applicability *function* $A(s)$*,*
- *Probability distributions* $P_a(s'|s)$*, for all* $s \in S$*,* $a \in A(s)$*,*
- *A* cost function $c(a, s)$*,* $c(a, s) > 0$ *for all* $s \in S$*,* $a \in A$*,*
- *A set of* observations Obs*,* goal signal $o_G \in Obs$*, and*
- *Probability distributions* $Q_a(o|s)$ *for* $a \in A$*,* $o \in Obs$*,* $s \in S$*.*

As with Goal MDPs M, goal states $s \in S_G$ are cost–free, $c(a, s) = 0$ for all a, absorbing, $P_a(s|s) = 1$ for all a, and *fully observable*, $Q_a(o_G|s) = 1$, when $s \in S_G$, and $Q_a(o_G|s) = 0$ otherwise. Goal *beliefs* are the beliefs b such that $b(s) = 0$ for $s \in S \setminus S_G$, in words, assign a probability of zero to non–goal states.

Discounted *cost–based* POMDPs (Sondik, 1978) differ from Goal POMDPs in two ways: goal states are not required and a *discount factor* $\gamma \in (0, 1)$ is used instead. While discounted POMDPs intuitively seem to be more expressive than Goal POMDPs , it has been shown that they can be compiled into Goal POMDPs following the transformation discussed in (Bonet and Geffner, 2009).

3.5 Goal POMDPs as MDPs over Belief Space

The most common way to solve POMDPs is by formulating them as completely observable MDPs over the *belief states* of the agent (Astrom, 1965; Sondik, 1978). Indeed, while the effects of actions on states cannot be predicted, the effects of actions on *belief states* can. More precisely, the belief b_a that results from doing action a in the belief b, and the belief b_a^o that results from observing o after doing a in b, are:

$$b_a(s) = \sum_{s' \in S} P_a(s|s')b(s')\,, \tag{3.5}$$

$$b_a(o) = \sum_{s \in S} Q_a(o|s)b_a(s)\,, \tag{3.6}$$

$$b_a^o(s) = Q_a(o|s)b_a(s)/b_a(o) \quad \text{if } b_a(o) \neq 0. \tag{3.7}$$

As a result, the *partially observable* problem of going from an initial state to a goal state is transformed into the *completely observable* problem of going from one *initial belief state* into a *goal belief state.* The Bellman equation for the resulting *belief* MDP is

$$V(b) = \min_{a \in A}\left\{c(a,b) + \sum_{o \in Obs} b_a(o)V(b_a^o)\right\} \tag{3.8}$$

for non–goal beliefs b and $V^*(b_G) = 0$ otherwise, where $c(a,b)$ is the expected cost $\sum_{s \in S} c(a,s)b(s)$, and b_G is a goal belief.

This belief MDP can be solved with a suitably modified version of the RTDP (Barto et al., 1995) or LRTDP (Bonet and Geffner, 2003) algorithm. The pseudo code for RTDP-Bel , a straight–forward modification of RTDP , is shown in Algorithm 3.2.

```
**Start** with $b = b_0$;
**Sample** state $s$ with probability $b(s)$;
**Evaluate** each action $a$ applicable on $b$ as
    $Q(a,b) = c(a,b) + \sum_{o \in O} b_a(o)V(b_a^o)$
initializing $V(b_a^o)$ to $h(b_a^o)$ if $b_a^o$ is not in the hash;
**Select** action $a$ that minimizes $Q(a,b)$;
**Update** $V(b)$ to $Q(a,b)$;
**Sample** next state $s'$ with probability $P_a(s'|s)$;
**Sample** observation $o$ with probability $Q_a(o|s')$;
**Compute** $b_a^o$ using Equation 3.7;
**Finish** if $b_a^o$ is goal belief;
**Else** $b := b_a^o$ and $s := s'$ and **goto** action evaluation step;
```

Figure 3.2: RTDP-Bel

RTDP-Bel is much like RTDP over a belief Goal MDP where states are replaced by belief states that are updated according to Equation 3.8, with an additional provision. When accessing the value $V(b)$ in the hash–table, b is replaced by $d(b)$, where d is the *discretization* function. Since b are probability distributions assigning real values in $(0,1)$ to states s, it is needed to bound the size of the hash table containing the values of the function $V(b)$. The discretization function function d proposed in (Bonet and Geffner, 2009) is extremely simple and maps each entry $b(s)$ into the entry:

$$d(b(s)) = \text{ceil}(D\,b(s))$$

where D is a positive integer, called the *discretization* parameter, and ceil(x) is the least integer $\geq x$.

Example 3.18 (Belief discretization)**.** *Let $b(s)$ be the vector*

$$b(s) = (0.22, 0.44, 0.34)$$

over the states $s \in S$. When $D = 10$, the discretized belief $d(b)$ *results in the vector*

$$d(b) = (3, 5, 4)$$

The discretized belief $d(b)$ is not a belief b and does not have to represent one faithfully. It just represents the unique cell in the hash table that stores the value function for b and of all other beliefs b' such that $d(b') = d(b)$. The discretization is used only to access the hash table representing $V(b)$ and it does *not* affect the beliefs that are generated during a trial. The discretization function d is a simple *function approximation* device (Sutton and Barto, 1998; Bertsekas and Tsitsiklis, 1996) where a single parameter, the value stored at cell $d(b)$ in the hash table, is used to represent the value of all beliefs b' that are discretized into $d(b') = d(b)$. The soundness of this approximation relies on the assumption that the value of beliefs that are close, should be close as well. Also important is that the discretization function preserves *supports* (Bonet, 2002), the states s with $b(s) > 0$, and thus never collapses the value of two beliefs if there is a state that it is excluded by one but not by the other, a property important in problems involving action preconditions.

While RTDP-Bel is a quite simple and effective algorithm, the discretization function used to represent the value function $V(b)$, entails several theoretical consequences (Bonet and Geffner, 2009). First, convergence is not guaranteed and actually the value in a cell may oscillate. Second, the value function approximated in this way does not remain necessarily a lower bound. These two severe shortcomings of RTDP-Bel can be addressed by storing in the hash table the value of a set of selected states s, and then using suitable interpolation methods for approximating the value for not stored states. This is what *grid–based* methods do (Hauskretch, 2000). These interpolations, however, involve a substantial computational overhead, and do not appear to be cost–effective (Bonet, 2002).

On the other hand, these limitations are common to most, if not all, of the linear or non–linear function approximation schemes used in practice (Sutton and Barto, 1998), and RTDP-Bel discretization is no exception. Moreover, these theoretical limitations do not seem to affect negatively the performance of RTDP-Bel when compared with Point–based algorithms for Goal Pomdps (Pineau et al., 2006), that are not subject to the limitations above. In practice, and over the benchmarks most commonly referred to in the Pomdp literature, RTDP-Bel and Lrtdp adapted for belief Mdps perform comparably to point–based schemes and sometimes are able to outperform those both in terms of the time to obtain a solution and the quality of the solution found (Bonet and Geffner, 2009).

3.6 GPT Modeling Language

Both Goal Mdps and Goal Pomdps are useful *models* for making explicit the mathematical structure of a wide class of planning problems with uncertainty. However, they are poor *languages* for describing them. This is due to the number and size of the relations and parameters involved. Like in classical planning models, where STRIPS and PDDL are used, it is preferable to describe problems compactly in terms of *modular*, *high–level*, factored languages. In the experiments we have carried out and described on Chapter 5 we have used the GPT planner (Bonet and Geffner, 2001b) which allows for this type of compact representation, and solves Goal MDP and Goal POMDP models with LRTDP (Bonet and Geffner, 2003).

As we did when discussing classical planning problems in Chapter 2, we will distinguish between the GPT POMDP *planning* model, where states and transition functions are defined implicitly, and the standard Goal POMDP model, where states and transition functions are defined explicitly:

Definition 3.19 (GPT Planning model)**.** *A* GPT *planning model is a tuple* $M = \langle F, A, I, G\rangle$ *whose elements are*

- *a set of* functional *fluents* F*,*
- *a set of* actions A*,*
- *a set of assignments to functional fluents* $f \in F$ *denoting the set of possible initial states,*
- *and a conjunction* G *of formulas over* F *denoting goal states.*

Depending on the definition of I and A, GPT induces a corresponding Goal MDP M, or Goal POMDP M_{Obs}. For example, if some of the actions are defined so that their effects are probabilistic and emit observation tokens, and I is a disjunction, the GPT pre–processor will derive automatically a Goal POMDP M'_{Obs} from the GPT planning model M.

Functions, Objects and Types

The GPT *language* is an extension of FUNCTIONAL STRIPS (Geffner, 2000), a typed first–order logical language that replaces the notion of *relational* fluents used in STRIPS by that of *functional* fluents. This allows the planner to avoid the requirement that objects be referred by their *unique* names, and results in a more compact representation. To illustrate this point, let's recall how the trainer location in the ROBOSCHOOL domain was encoded in STRIPS:

$$\text{trainerAt}(l)$$

where l was a variable symbol to be replaced during grounding by some object of type *location*. In FUNCTIONAL STRIPS fluents *trainerAt*(l) would be replaced by *one* single functional fluent

$$\text{trainerAt}()$$

which would denote a 0–arity function, whose domain would be that of objects of type *location*. While this difference seems to be mere syntactic sugar, it results in much more succinct representation.

In the STRIPS representation, the size of the state space S is *proportional* to $2^{|locations|}$, since each fluent would account for the fact that the trainer is at some determinate location l, and these facts can be either true or false in any state $s \in S$. FUNCTIONAL STRIPS allows for a more succinct interpretation, since the underlying state space is proportional to $|locations|$ since the functional fluent trainerAt() can only have *one* possible value.

STRIPS and FUNCTIONAL STRIPS are syntactically and, to some extent, semantically similar. However, modeling tasks in GPT in such a way that the greater succinctness of the latter is exploited requires abandoning common STRIPS modeling practices. For instance, in the ROBOSCHOOL task, we had *pieces* objects the trainer agent could interact with. These objects were decorated in a number of ways, namely, color, shape and location. STRIPS common practice would be to model these as we did in Table 2.2, introducing binary predicates, whose arguments would correspond to the *name* of the piece and the *value* of that attribute. In FUNCTIONAL STRIPS we define these as functional symbols

$$\begin{aligned} \text{location} &: Piece \rightarrow Location \\ \text{color} &: Piece \rightarrow Color \\ \text{shape} &: Piece \rightarrow Shape \end{aligned}$$

where *Piece*, *Location*, *Color* and *Shape* refer to the types – sets of constant symbols – with the same name, rather than as *binary* functions whose domain are boolean constants *true* and $false$. Another example of a FUNCTIONAL STRIPS fluent for the ROBOSCHOOL situation in Figure 1.1 would be the formula:

$$location(p_1) = A0 \tag{3.9}$$

which states that the location of piece p_1 is cell $A0$. Note the difference with the relational fluent $location(p_1, A0)$, that requires the object $A0$ to be explicit in the fluent signature. Constant fluent symbols can be encoded as function symbols of arity 0, while relational fluent symbols can be encoded as function symbols of the same arity plus equality.

Functional symbols can be combined to denote more complex formulas, using the standard logic connectives $\wedge$, $\vee$, $\neg$, as well as equality $=$. Besides these, functional symbols that map objects to the set of integers can be connected with the relational operators $<$, $>$, $\leq$ and $\geq$.

Given FUNCTIONAL STRIPS together with the relevant type and object declarations, the *states* are then *logical interpretations* over this language. That is, a state s assigns a *denotation* x^s to any symbol x from which the denotations of atoms, terms and formulas are obtained following the standard composition rules. Certain symbols, such as $A0$, however, have a denotation that is fixed and independent of the state. States thus have to assign a denotation to *fluent* symbols like *location*. *Type* and *object* declarations for these symbols define possible sets of denotations, or values, and all together *implicitly* define the state space S.

Actions

FUNCTIONAL STRIPS model of agent actions $a = (Pre, Eff, Obs, c)$ consists of four elements: a *precondition* formula Pre, a set of statements denoting action a *effects* Eff, *observation* tokens Obs emitted and a cost $c \geq 0$.

Preconditions can be any valid formula over functional symbols F. They implicitly define the set of applicable actions to some state s since the only actions a executable are those whose preconditions have a true denotation in s.

Action **effects** depart significantly from those of STRIPS actions. In FUNCTIONAL STRIPS action postconditions are sets σ_i of statements of the form:

$$f(t) := w$$

that is, assignments of values – constant symbols – w to functional fluents f defined over objects t.

To highlight the differences between STRIPS and FUNCTIONAL STRIPS consider the $move(A0, B0)$ action from the ROBOSCHOOL example as a modeled in STRIPS:

$$\begin{aligned} Pre &: \text{trainerAt}(A0) \\ Add &: \text{trainerAt}(B0) \\ Del &: \text{trainerAt}(A0) \end{aligned}$$

becomes in FUNCTIONAL STRIPS

$$\begin{aligned} Pre &: \text{trainerAt}() = A0 \\ Effects &: \text{trainerAt}() := B0 \end{aligned}$$

The most important difference lies in how action effects are specified. While in STRIPS it is required to state explicitly that the precondition trainerAt($A0$) is no longer true, in FUNCTIONAL STRIPS this is implicitly stated when the value assigned to the symbol trainerAt() is changed.

GPT extends FUNCTIONAL STRIPS by adding to action effects the notion of *ramification* rules. These rules allow one to specify several possible outcomes of the execution of an action a or to specify indirect effects of an action. Two kinds of ramification rules are supported: *conditional* effects and *probabilistic* effects.

The former, denoted (x_i, σ_i), are the exact equivalent of ADL conditional effects (Anderson et al., 1998). The conditional effect (x_i, σ_i) will only affect the state resulting from executing an action if x_i is true in the state s where the action is applied. GPT probabilistic effects, denoted (p_i, σ_i), become true after action execution with some probability p_i. If an action has n probabilistic effects (p_i, σ_i) then $\sum_{i=1}^{n} p_i$ must indeed be 1. A valid action description must have at least 2 probabilistic effects.

A natural example that arises in the ROBOSCHOOL task of the usage of ramification rules is to consider the throw() action to have two probabilistic effects. One where the piece thrown does actually fall into the target bin with probability p, and with probability $1 - p$ that it falls outside but still near to the bin.

When a GPT planning model M does not contain any action $a \in A$ with probabilistic effects, the transition probability distribution P_a is indeed *deterministic*, so the state

s' resulting from executing action a in state s is like the latter, but replacing symbol values by the new values prescribed in a's effects. When probabilistic effects are present, s' is a random variable with domain S_a, the set of states s_i resulting from setting symbols $x \in \sigma_i$ to the new values prescribed.

For describing Goal Pomdps, it is necessary to further extend Functional Strips to be able to describe what is *observable*, implicitly defining the sensor model $Q_a(o|s)$. Gpt allows specifying **observation tokens** as formulas over F, attaching to action descriptions fields $o : (x = v)$ for each possible denotation v of x with different probabilities that depend on the belief state where the action was done and the sensor model as described by Equations 3.5–3.7.

For instance, if in the RoboSchool task, the location of some pieces was not known to the trainer beforehand, we could have a seek(p) action and the observation would be encoded by the following formula:

$$location(p) = trainerAt()$$

This action would emit *two* different tokens, one corresponding to the case when the formula above is true in s and the other corresponding to the case when the formula is false.

Initial Beliefs and Goal States

Initial beliefs are specified by a set of assignments of values w to functional symbols, very much like action effects. Gpt extends Functional Strips by introducing language constructs that specify sets of equally likely values for functional symbols in F, so that the initial belief b_0 becomes an actual probability distribution over S. If possible states are not equally probable, then it is required to introduce a *start* action with probabilistic effects that correspond to each possible initial state.

Goal states are denoted as *conjunctions* of Functional Strips formulas.

3.7 Summary

This chapter has introduced two models that allow more realistic representations of agents and environments, by relaxing the restrictive assumptions regarding deterministic action outcomes and full–observability made in classical planning. The computational problem posed by solving these two models is also substantially harder, and existing algorithms for this task have trouble scaling up as neatly as classical planners do over many domains. One of the reasons for this shortcoming lies in the inability of existing algorithms to effectively analyze factored representations and propagate updates to $Q(a, s)$ values so these simultaneously affect many states sharing common features. The other reason lies in the inability to avoid processing those parts of the state or belief state space which are not relevant or necessary to consider in order to find an optimal solution. These two abilities, present in classical planners through domain–independent heuristics and search algorithms operating on a implicitly defined search space, would significantly speed up the computation of solutions. Improving scalability on Mdps and Pomdps is an active field of research (Kolobov et al., 2009; Silver and Veness, 2010) and we expect the current situation to change rapidly over the next few years.

Part II

Planning–based Plan Recognition

Chapter **4**

PR over Classical Planning Theories

This chapter starts presenting a *crisp* definition of plan recognition problems in terms of Strips planning domains and their solutions. Solutions to plan recognition problems are functions defined over the costs of optimal plans for the underlying classical planning problems defined by a set of *hypothetical* goals that comply with the observed action sequence. We then discuss the goal filtering scheme first presented in (Ramirez and Geffner, 2009) and its limitations. These are addressed slightly reformulating plan recognition problems to accomodate a prior probability distribution over hypothetical goals. Solutions then become a well–founded posterior goal distribution $P(G|O)$ over hypothetical goals G. This distribution is founded on the relationship between costs of optimal plans satisfying the provided observation sequence and those which do not (Ramirez and Geffner, 2010). After presenting the theoretical and practical contributions of our framework for plan recognition, we finish the chapter by introducing several plan recognition tasks we use to evaluate our formulation for plan recognition. These are either derived from well–known benchmarks used by the planning community or are the result of mapping plan libraries, discussed in the plan and action recognition literature, into suitably defined planning problems. The chapter ends with a detailed explanation of the experimental setup used to evaluate our formulation and the results of optimal and approximate algorithms to compute $P(G|O)$.

4.1 Preliminary Definitions

We start defining the plan recognition problem:

Definition 4.1 (Plan recognition problem)**.** *A plan recognition problem or* theory *is a triplet $T = \langle P[\cdot], \mathcal{G}_T, O\rangle$ composed by*

- *A* planning domain $P[\cdot] = \langle F, A, I\rangle$,
- *a* set of possible goals $\mathcal{G}_T$, *each $G \in \mathcal{G}_T$ and $G \subseteq F$,*
- *an* observation sequence $O = (o_1, \ldots, o_n)$, *where each $o_i \in A$.*

The planning domain $P[\cdot]$ implicitly encodes the behavior the observed agent may engage into, for any goal expressed as a subset of fluents F. Possible goals are circumscribed to those explicitly stated in $\mathcal{G}_T$. For a goal $G \in \mathcal{G}_T$, the set of hypothetical goals, $P[G]$ defines a complete classical planning problem.

Observation sequences O are *totally ordered* sequences of actions specified in the planning domain. It is not required for actions in O to be contiguous actions in some plan, nor to O to be a plan *prefix*. For example an agent following the plan

$$\pi = (a, b, c, d, e, f)$$

may generate the following observation sequences: $O = (b, d, f)$, $O = (a, f)$ and indeed $O = (a, b, c)$.

In general, some of the valid plans π for planning problems $P[G]$, $G \in \mathcal{G}_T$, will not account for the observed action sequence O. For example, the plan

$$\pi = (a, b, d, a, c, a)$$

satisfies the observation sequences $O_1 = (a, d, a)$ and $O_2 = (b, a, a)$, but does not satisfy sequence $O_3 = (c, b, d)$. In words, there is in π an occurrence of each of the actions in O_1 and these occur in the *same* order, e.g. a before d and d before a. This can be formalized by introducing a function that maps observation indices in O into action indices in π.

Definition 4.2 (Plan π satisfying observation sequence O)**.** *A plan* $\pi = (a_1, \ldots, a_n)$ *satisfies the observation sequence* $O = (o_1, \ldots, o_m)$ *if there exists a strictly* monotonic *function* f *mapping the observation indices* $i = 1, \ldots, m$ *into action indices* $j = 1, \ldots, n$ *such that* $a_{f(i)} = o_i$.

For example, the unique function f that establishes a correspondence between the actions o_i observed in O_2 and the actions a_j in π is $f(1) = 2$, $f(2) = 4$ and $f(3) = 6$. This function must be strictly monotonic[1] so that the action sequences π preserve the ordering of the actions observed. On the other hand, no such function f exists for the observation sequence $O_3 = (c, b, a)$, since there is no occurrence of c in π with an index lower than 2, which is the position of the only occurrence of b. As a result π satisfies O_2 but not O_3.

Existence of plans π satisfying O, for one of the problems $P[G]$ conveyed by T, is indeed *sufficient* to determine that G is relevant for explaining O. However, the mere existence of such plans does not allow to leverage how adequate or contrived is the explanation of O conveyed by goal G. Obviously, if one finds that there is one single goal G in our hypothetic goal set $\mathcal{G}_T$ admitting valid plans π that satisfy O, solving T would amount to select G as the actual agent goal. This is rarely the case in many plan recognition problems.

Let us see what goals have plans that satisfy the observations in the RoboSchool task depicted on Figure 1.1. In this case we observe the trainer to pick up the red triangle first and then the blue triangle – objects p_3 and p_4. Since there are plans π_i satisfying O for each of the goals $G_i \in \mathcal{G}_T$, we would end up considering *all* the goals in the $\mathcal{G}_T$ set as *equally good* explanations of the observed actions, since every goal

[1]A function is *monotonic* if for any two $x > y$, $f(x) > f(y)$.

G in $\mathcal{G}_T$ admits a valid plan which would satisfy that observation sequence. On the other hand, if the observation sequence was just to see the trainer to leave its initial position – cell $E0$ – and move into cell $D0$, then the existence criterion would select as well all the hypothetic goals $\mathcal{G}_T$ and this would indeed be reasonable, since *all* plans for any of the goals would *necessarily* contain that action.

4.2 Accounting for Rational Agents

Finding a plan for $P[G]$ satisfying O does tell us that G is a *possible* goal when O is observed, but does not really tell us to what extent satisfying O would be a *necessary* property of plans achieving goal G. This is precisely the gist of the objections raised by (Kautz and Allen, 1986a; Charniak and Goldman, 1993) on earlier approaches to plan recognition (Schmidt et al., 1978; Perrault and Allen, 1980) which relied on the possibility to find plans that matched observations O as the *only* reason for deeming a goal G as a possible explanation of O.

We note that one can relate the property of plans satisfying observation sequences O, and that of O being necessary for achieving G through another property of plans: their *cost*. Inferring the goals behind other agents' observed behavior consists in showing that there is a causal relation between the observed agent knowledge and goals and the plans they execute to achieve those goals. (Dennett, 1983) convincingly argues that such a causal relationship should be governed by the *principle of rationality*. That is, the expectation that intentional agents will tend to choose actions that achieve their goals *most efficiently*, given their knowledge of the actual world state. A STRIPS planning domain augmented with a cost function captures well the notion of *efficiency* and *optimal* plans[2] achieving a goal G set the benchmark by being indeed the *most* efficient plans for that goal.

Plan cost functions and optimal plans allow us to specify to what degree a plan satisfying O *deviates* from the most efficient – optimal – plan for a goal G. This constitutes a crisp criterion to leverage how necessary is to come up with a plan that satisfies some given observation sequence O in order to achieve a goal G in the most efficient possible manner. Therefore, the notion of *rationality* in our formulation of plan recognition corresponds with that of agents seeking and pursuing optimal plans for a given planning problem $P[G]$. Plan recognition system designers should be aware that the cost function they *expect* to be minimized by the observed agent encoded in the cost function c, might possibly be different from that the observed agent *actually* cares about.

Since cost of optimal plans is a property of planning problems $P[G]$, or more precisely, of *goals* alone, then it becomes apparent that we can *rank* goals $G \in \mathcal{G}_T$ by comparing the minimum cost of plans satisfying O.

Definition 4.3 (Minimum cost of satisfying plans O)**.** *The minimum cost of plans π that satisfy observation sequence O for problem $P[G]$ is defined as*

$$c^*_P(G,O) = \min_{\pi_O \in \Pi_P(G,O)} c(\pi)$$

[2]See Definition 2.8, where we introduce the notion of cost functions in STRIPS planning, and Definition 2.9, where *optimal plans* are defined.

where $\Pi_P(G, O)$ is the set of all valid plans π_O for $P[G]$ that satisfy O, and $\Pi_P(G, O) \subseteq \Pi_P(G)$, the set of all valid plans for $P[G]$.

with the minimum cost of achieving G regardless of O:

Definition 4.4 (Minimum cost difference $\Delta(G, O)$)**.** *The* deviation *of plans for $P[G]$ from optimal solutions to the planning problem $P[G]$ when satisfying O, $\Delta(G, O)$ is*

$$\Delta(G, O) = |c_P^*(G, O) - c_P^*(G)|$$

so a relationship is established between the lower bound on the cost of plans that achieve G and the lower bound on the cost of plans that both achieve G and satisfy O.

4.3 A Qualitative Model of Plan Recognition

With these definitions in place, we are ready to offer a definition of *qualitative* solutions to plan recognition theories T.

Definition 4.5 (Optimal goal set)**.** *The* qualitative *solution to a theory $T = \langle P, \mathcal{G}_T, O\rangle$ consists in computing the* optimal goal set *$\mathcal{G}_T^*$ which comprises the goals $G \in \mathcal{G}_T$ such that $\Delta(G, O) = 0$.*

The solution to a plan recognition theory T is given then by the hypothetic goals $G \in \mathcal{G}_T$ that admit an optimal plan that is compatible with the observations, or in other words, the goals G for which there is no cost increase in the plans that achieve G when O must be satisfied.

In order to illustrate the discussion in this Section, we will refer to the example plan recognition theory depicted in Figure 4.1. Out of the six goals in $\mathcal{G}_T$ only two have optimal plans that satisfy the observation sequence

$$O = (walk(C0, B1), pick(p_3, A4))$$

These goals are G_1, "store all triangles into b_1" and G_4, "store all red objects into b_2", since $\Delta(G_1, O) = 0$ and $\Delta(G_4, O) = 0$. For the other four goals, while they all have valid plans that satisfy O, these have a cost higher than the optimal cost to achieve them.

For instance, for goal G_2, "store all spheres into b_2", the deviation when satisfying O, $\Delta(G_2, O)$, is $2 + 3\sqrt{2}$, because of the additional actions that require to be executed in order to comply with the observed actions in O. Table 4.1 shows the optimal plan for achieving G_2 and the optimal plan for achieving that same goal while satisfying O.

This example illustrates the intuition behind the plan recognition schemes discussed in Sections 4.5 and 4.7, that of *plan costs* being the criterion that follows from the assumed rationality of the observed agent. If the agent is to prefer efficient plans over those less efficient, then goals whose most efficient plans that satisfy the observation sequence O are found to have a cost higher than that of optimal plans, are not an explanation as good as those goals where satisfying O does not entail executing a worse quality plan. Next section explains how plans which satisfy or do not satisfy O can be efficiently computed.

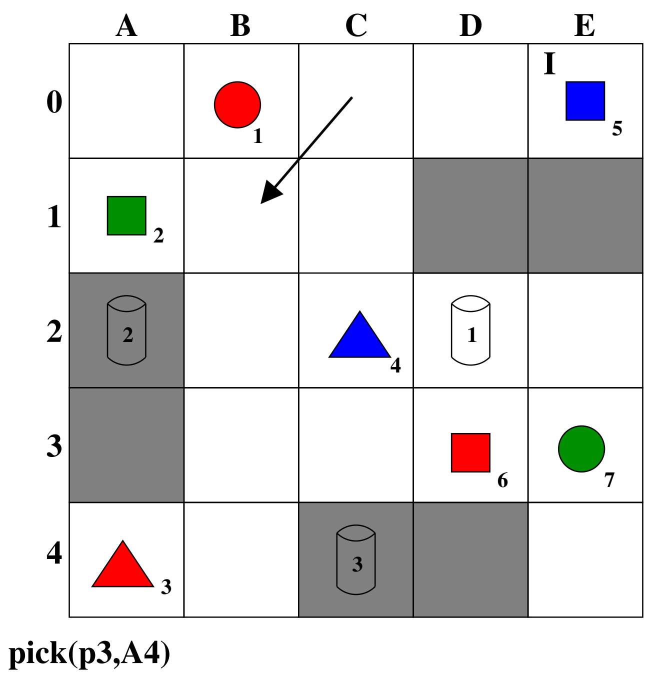

Figure 4.1: A RoboSchool plan recognition theory. Planning domain $P[\cdot]$ and hypothetic goal set $\mathcal{G}_T$ are the same as those described on Section 2.10. The trainer starts at cell $E0$, marked with a capital I letter. Observation sequence O consists of two actions: $walk(C0, B1)$, shown by an arrow between those two cells, and $pick$ $(p_3, A4)$, displayed by the text next to the cell in question.

4.4 Computing Plans that Satisfy Observations

In this section we present a scheme to compute optimal plans that satisfy observation sequences O with an off–the–shelf classical planner, that is, without modifying its code in any way. We will show that obtaining costs of plans for goals G that satisfy, or are guaranteed to *not* satisfy, an observation sequence O, amounts to compute the costs of plans for slightly different goal G', $G \subset G'$. This requires to add to the original planning domain $P[\cdot]$ fluents that account for observations in O being embedded in the proper order in the plan, as well as additional actions, which are like original actions in $P[\cdot]$ but with the addition of the fluent corresponding with an observation *only* when all previous observations have been already accounted for.

So rather than tweaking the planner or its heuristic as in previous approaches to plan recognition using planners (Schmidt et al., 1978; Perrault and Allen, 1980), we derive from each of the planning problems $P[G]$ present in a plan recognition theory T, planning problem $P'[G+O]$ ($P'[G+\overline{O}]$), where $G+O$ ($G+\overline{O}$) is the modified goal formula, whose valid plans, either optimal or otherwise, are guaranteed to satisfy (not satisfy) observation sequence O.

For simplicity in the presentation, we will assume that no pair of observations o_i and o_j in O refer to the same action a in P. When this is not the case, we can create a copy a' of action a so that o_i refers to a' and o_j refers to a.

G_2	G_2 *and* O
walk($E0, D0$)	walk($E0, D0$)
walk($D0, C0$)	walk($D0, C0$)
walk($C0, B0$)	walk($C0, B1$)
pick($i_4, B0$)	walk($B1, B0$)
walk($B0, B1$)	pick($i_4, B0$)
throw($i_4, B1, b_2, A2$)	walk($B0, B1$)
walk($B1, C2$)	throw($i_4, B1, b_2, A2$)
walk($C2, D3$)	walk($B1, C2$)
walk($D3, E3$)	walk($C2, D3$)
pick($i_7, E3$)	walk($D3, E3$)
walk($E3, D3$)	pick($i_7, E3$)
walk($D3, C3$)	walk($E3, D3$)
walk($C3, B3$)	walk($D3, C3$)
throw($i_7, B3, b_2, A2$)	walk($C3, B3$)
	walk($B3, A4$)
	pick($i_3, A4$)
	walk($A4, B3$)
	throw($i_7, B3, b_2, A2$)
$11 + 2\sqrt{2}$	$13 + 5\sqrt{2}$

Table 4.1: Two valid plans for $P[G_2]$. On the left, an *optimal* plan achieving goal G_2 is shown, which does not satisfy the observation sequence $O =$ (walk($C0, B1$), pick($p_3, A4$)). On the right, we see the minimum cost plan that both satisfies O and achieves G_2. Bottom row values show costs of both plans.

Mapping observations into actions

We will compile the observations away by mapping the theory $T = \langle P, \mathcal{G}_T, O\rangle$ into an slightly different theory $T' = \langle P', \mathcal{G}_T, O'\rangle$ with an empty set O' of observations. P' is like P but for the actions a corresponding with observations $o \in O$:

Definition 4.6 (Mapping observations into actions)**.** *For a domain $P = \langle F, I, A\rangle$ and the observation sequence O, the new domain is $P' = \langle F', I', A'\rangle$ with*

- $F' = F \cup \{p_a \mid a \in O\}$
- $I' = I \cup \{\neg p_a \mid a \in O\}$
- $A' = A$

where p_a is a new fluent and the actions $a \in A'$ that are in O have an extra effect:

- $\emptyset \rightarrow p_a$ *when a is the* first *action in O*
- $p_b \rightarrow p_a$ *when b is the action that* immediately precedes *a in O*

The mapping above requires to extend Strips with two language features that enhance its expressiveness and allow to represent planning problems in a more succint way (Pednault, 1989). One are *conditional effects* [3] so $p_b \rightarrow p_a$ prescribes that p_a is

[3]Already introduced in Section 2.2.

added to the state s' resulting from applying the action *only* when p_b was already true in the state s the action is being applied. The second is *negation*, which allows to handle explicitly the fact that a certain observation $b \in O$ has not been already accounted for, i.e. p_b is *false* in some state s. Both extensions to STRIPS can be compiled into regular STRIPS representations (Gazen and Knoblock, 1997).

The purpose for the mapping above is that we want the planner to compute, in *parallel*, plans and satisfiability of O, without either computation interfering in any way with the other. In order to do so, it suffices to add the fluents p_a and modify actions in the manner described by Definition 4.6. Adding the fluents p_a and modifying actions as per Definition 4.6, checking whether a plan π satisfies O becomes a *side effect* of plan execution, so its *implicitly* computed by a standard classical planner. In the transformed domain P', the state s' resulting of applying plan π on the initial situation I, not only encodes the world situation after plan execution, but also encodes *explicitly* whether the plan is satisfying the observation sequence O or not.

This allows to partition implicitly the set of valid plans for $P[G]$ into two distinct sets. Namely, the set of plans that satisfy O, so the fluent p_a corresponding to the last observed action a in O is true after these plans are executed. Conversely, the set of plans that do not satisfy O is also implicitly defined to be those plans that do not make this fluent true after its execution.

Partitioning the Space of Valid Plans

The compilation procedure in Definition 4.6 splits the set of valid plans for a goal $\Pi_P(G)$ into two disjoint subsets corresponding with the sets of plans that satisfy, or do not satisfy, the observation sequence O. This split is achieved implicitly by altering the planning domain $P[\cdot]$ so the semantics of $P[G]$, the set of valid plans for goal G identity, and the costs of such plans are preserved.

We start by introducing goals $G + O$

$$G + O = G \cup \{p_b\}$$

and $G + \overline{O}$

$$G + \overline{O} = G \cup \{\neg p_b\}$$

where p_b is the *last* action in observation sequence O. Plans for each of these two goals correspond exactly to plans that respectively satisfy or do not satisfy O, as we show with the following theorems.

Theorem 4.7. *Let $\pi = (a_1, \ldots, a_n)$ be a plan for $P[G]$ that satisfies $O = (o_1, \ldots, o_m)$. Then the goal $G + O$ is true after the execution of π on initial state I', $G \cup \{p_{b_m}\} \in I'[\pi]$.*

Proof. We observe that G will be true after the execution of π on I' since the transformation in Definition 4.6 does not modify in any way actions precondition of unconditional effects. We will prove that $p_{b_m} \in I'[\pi]$ when π satisfies O by induction.

We recall from Definition 4.2 that when π satisfies O, then exists f, monotone mapping between indices $[1, m]$ corresponding to actions in O and $[1, n]$ indices of actions in π, such that for every $b_j \in O$, $f(j) = i$ and $a_i = b_j$, and that for any $b_k \in O$, $k > j$, then $f(k) > f(j)$.

It is easy to see that $p_{b_1} \in I'[a_1, \ldots, a_{f(1)}]$, since $a_{f(1)} \in A'$ has conditional effect $\emptyset \rightarrow p_{b_1}$ which unconditionally adds fluent p_{b_1}. We assume to be true that $p_{b_j} \in I'[a_1, \ldots, a_{f(j)}]$ since conditional effect $p_{b_{j-1}} \rightarrow p_{b_j}$ has been triggered. In order to show that $p_{b_{j+1}} \in I'[a_1, \ldots, a_{f(j+1)}]$ we rely on a proof by contradiction.

If $p_{b_{j+1}} \notin I'[a_1, \ldots, a_{f(j+1)}]$ then the conditional effect $p_{b_j} \rightarrow p_{b_{j+1}}$ associated with action $a_{f(j+1)}$ has not been activated. Since $p_{b_j} \in I'[a_1, \ldots, a_{f(j)}]$, and given that $f(j) < f(j+1)$ by Definition 4.2 and there is no action $a' \in A'$ deleting fluent p_{b_j} then for any action $a_k \in \pi$, we know that $p_{b_j} \in I'[a_1, \ldots, a_k]$, with $f(j) < k < f(j+1)$.

This contradicts the assumption that $p_{b_{j+1}} \notin I'[a_1, \ldots, a_{f(j+1)}]$, since conditional effects are *necessarily* triggered when the their conditions are entailed by the state where action they are associated to is being executed. Therefore we can conclude that $p_{b_j} \in I'[a_1, \ldots, a_{f(j)}]$ for all j, $1 \leq j \leq m$ when π satisfies O. □

We next show that plans π for $P'[G + O]$ correspond to plans for $P[G]$ that satisfy O:

Theorem 4.8. *Let $\pi = (a_1, \ldots, a_n)$ be a plan for $P'[G + O]$. Then π is a plan for $P[G]$ that satisfies $O = (o_1, \ldots, o_m)$ as per Definition 4.2.*

Proof. If π is a plan for $P'[G + O]$ then $G \subseteq I[\pi]$, since actions $a' \in A'$ have each a matching action $a \in A$ with the same precondition and unconditional effects. In order to π to satisfy O there must exist f, monotone mapping of indices $[1, m]$ of actions in O into indices $[1, n]$ of actions in π. Such a mapping can be obtained by defining f as follows

$$f(i) = \max\{i \mid p_{b_j} \in Adds(s, a_j),\ a_j \in \pi\}$$

We know f to be monotone since plans for $P'[G+O]$ implicitly define such a mapping as shown on Theorem 4.7. □

The correspondence between plans for $P[G]$ that do not satisfy O and plans for $P'[G + \overline{O}]$ is guaranteed by the following Theorem:

Theorem 4.9. *Let $\pi = (a_1, \ldots, a_n)$ be a plan for $P[G]$ that does not satisfy $O = (o_1, \ldots, o_m)$. Then π is a plan for $P'[G + \overline{O}]$.*

Proof. In order to show that π is a plan for $P'[G + \overline{O}]$ it suffices to see that

$$G \cup \{\neg p_{b_m}\} \subset I'[\pi]$$

We recall that $\neg p_{b_m} \in I'$, so in order to π to be a plan for $P'[G + \overline{O}]$ it is required to show that $\neg p_{b_m}$ will not be deleted by any action $a \in A'$ in the plan π. We will prove that this is indeed the case by contradiction.

Let a_k be an action in plan π, where $1 \leq k \leq n$, conditional effect $p_{b_{m-1}} \rightarrow p_{b_m}$. In order to p_{b_m} to be asserted when executing a_k, it is required that there exists some action $a_l \in \pi$, such that $1 \leq l < k$, with conditional effect $p_{b_{m-2}} \rightarrow p_{b_{m-1}}$. It is easy to see that this eventually lead to $p_{b_1} \in I'[\pi]$, and by extension, to the existence of a monotone mapping f between indices of actions in O and π. This, in turn, would mean that π does actually satisfy O, which contradicts our earlier statement of π not doing so. □

Finally we need to see that plans π for $P'[G + \overline{O}]$ are plans for $P[G]$ that do not satisfy O:

Theorem 4.10. *Let $\pi = (a_1, \ldots, a_n)$ be a plan for $P'[G + \overline{O}]$. Then π is plan for $P[G]$ that does not satisfy $O = (o_1, \ldots, o_m)$.*

Proof. Let f be the following mapping between $[1, m]$, indices of actions in O, and $[1, n]$, the indices of actions in plan π:

$$\begin{aligned} f : \quad & [1, m] \rightarrow && [1, n] \\ & j \mapsto && j,\ a_i = b_j,\ a_i \in \pi,\ b_j \in O \end{aligned}$$

Since $\neg p_{b_m} \in I'[\pi]$ either one of the following must be true:

1. There exists some $b_j \in O$ such that $f(j)$ is not defined.
2. There exist actions $b_i,\ b_j \in O$, with $1 \leq j \leq i \leq m$ such that $f(j) > f(i)$.

Either condition prevents a monotone mapping f to exist between actions in O and π, so we can conclude that π indeed does not satisfy O. □

The previous results can be summarized in the following Corollary:

Corollary 4.11 (Correspondence between plans)**.** *Let $G+O$ stand for goal $G \cup \{p_a\}$ and $G+\overline{O}$ for goal $G \cup \{\neg p_a\}$, where a denotes the* last *action in observation sequence O. The following correspondence between plans exist:*

1. *π is a plan for $P[G]$ that satisfies O iff π is a plan for $P'[G + O]$*
2. *π is a plan for $P[G]$ that does* not *satisfy O iff π is a plan for $P'[G + \overline{O}]$.*

The following two Theorems state that the deviation from optimal behavior $\Delta(G, O)$ from Definition 4.4 is preserved for transformed domains $P'[\cdot]$. First, cost of plans is not altered by the transformation described by Definition 4.6:

Theorem 4.12 (Preservation of plan costs in $P[\cdot]$ and $P'[\cdot]$)**.** *Let $P[\cdot] = \langle F, I, A \rangle$ be a planning domain, $G \subseteq F$ an arbitrary subset of the fluent set F, and π be a plan for $P[G]$. Let π be a plan for $P[G]$. Let π' be a plan for problem $P'[G]$, where $P'[\cdot]$ is the result of applying Definition 4.6 to $P[\cdot]$. Then it holds that*

$$c^*(\pi) = c^*(\pi')$$

Proof. As stated by the Theorem 4.11 there is a surjective mapping between plans π in the set $\Pi_P[G]$ and those plans π' in the set $\Pi_{P'}[G + O] \cup \Pi_{P'}[G + \overline{O}]$. The number of actions in π and π' is *not changed.* Therefore, since Definition 4.6 does not prescribe any change in the cost function associated with $P[\cdot]$, the cost of π and that of π' is the same. □

So $\Delta(G, O)$ can be formulated directly as the difference between the optimal costs of plans for goals $G + O$ and G in the transformed domain $P'[\cdot]$:

Theorem 4.13 ($\Delta(G, O)$ preserved by Definition 4.6)**.** *For any goal G and observation sequence O, it holds that:*

1. $c^*_P(G, O) = c^*_{P'}(G + O)$
2. $c^*_P(G) = c^*_{P'}(G)$

where c^*_P *and* $c^*_{P'}$ *are respectively the optimal costs of* $P[G]$ *and* $P'[G]$*,* $P'[\cdot]$ *denotes the planning domain that results from applying Definition 4.6 on planning domain* $P[\cdot]$*. As a result the* deviation $\Delta(G, O)$ *from optimal solutions to the planning problem* $P[G]$ *when satisfying O becomes*

$$\Delta(G, O) = |c^*_P(G, O) - c^*_P(G)| = |c^*_{P'}(G + O) - c^*_{P'}(G)|$$

Proof. Since costs of plans in $P'[\cdot]$ do not change with respect to their cost in $P[\cdot]$ from Theorem 4.12 then the *minimum* cost of any set of plans does not either change. □

Once presented the formulation for obtaining plans that satisfy observation sequences O and having shown it to be correct, we now discuss the implications of the mapping given in Definition 4.6 with respect to the efficiency of plan search.

Efficiency

In general, compilation of conditional effects implies an *exponential* increase in the number of actions in the STRIPS representation. This blow–up depends on the number of fluents in the *antecedent* of a conditional effect. As it is obvious, in the case of the mapping above, the antecedent size is bounded by a constant, and this constant is 1. In effect, planners not supporting natively conditional effects will be handling a domain with *twice* as many operators.

On the other hand, adding fluents p_a also potentially increases the number of states S in the state space Π induced by planning domain $P[\cdot]$ from $2^{|F|}$ to $2^{|F|+|O|}$, that is, $2^{|O|}$ times bigger. However, the number of *reachable* states, which are the states which can *actually* be reached from initial state I during plan search, does not. It is easy to see that there are just $|O|$ times more reachable states:

Theorem 4.14 (Reachable states increase)**.** *Let* $P'[\cdot]$ *be the planning domain resulting from applying Definition 4.6 to some planning domain* $P[\cdot]$*. The number of* reachable *states in* $P'[\cdot]$*, for any goal G, is* $|O|$ *times the number of reachable states in* $P[\cdot]$*.*

Proof. Let $s \subseteq F'$ be a state where fluent p_a is true and fluent p_b is false, a and b being actions in O, and b comes *before* a in O. This state cannot be reached from I', since in order for p_a to be true, p_b needs to have been made true *at some point* by some action with conditional effect $p_c \rightarrow p_b$, c being an action preceding both a and b in O, or an action with conditional effect $\emptyset \rightarrow p_b$ if b is the *first* action in O. In order to p_b and p_a to be simultaneously false and true, there must exist some action *d deleting* p_b. But there does not exist any action in $P'[\cdot]$ doing so. Hence, we can conclude that there is no state s reachable from I' where there are fluents p_a and p_b being respectively true and false *and* b being an action preceding a in O. □

Since O is *totally ordered*, and this is enforced by the conditional effects antecedents and the semantics of STRIPS actions, there are many possible combinations of true (false) fluents p_a as actions in O plus one.

4.5 Computation of Optimal Goal Set

Now we have all the components needed to propose an algorithm for solving plan recognition theories T that relies on classical planners:

Proposition 4.15 (Computation of $\mathcal{G}_T^*$)**.** *Let $T = \langle P, \mathcal{G}_T, O\rangle$ be a plan recognition theory, and $T' = \langle P', \mathcal{G}_T, O'\rangle$ be the theory resulting the transformation described in Definition 4.6 to T. Then $\mathcal{G}_T^*$ can be computed as:*

$$\mathcal{G}_T^* = \{G \,|\, G \in \mathcal{G}_T,\, \Delta(G, O) = 0\}$$

that requires $2 \cdot |\mathcal{G}_T|$ calls to an optimal *classical planner to obtain $c^*_{P'}(G + O)$ and $c^*_{P'}(G)$.*

Computing $\mathcal{G}_T^*$ with the procedure outlined above provides us with a *qualitative* solution to a plan recognition theory. That is, the set of goals that are found to be *consistent* with the assumption that the observed agent tries to achieve its goals in the most efficient manner. Goals G with $\Delta(G, O) > 0$ are a worse explanation for O than goals G' with $\Delta(G', O) = 0$, as the former are not consistent with this assumption. If G was to be the actual goal, the fact that best plan supporting O has worse quality than a plan that does not support it, would point to the agent *not* behaving in the most cost efficient possible way. This qualitative notion intuitively appeals to the probabilistic notion of the likelihood of being O observed when G is being pursued.

4.6 Limitations of Qualitative Model

For plan recognition problems such as the one shown in Figure 4.1, the procedure for computing $\mathcal{G}_T^*$ given on Definition 4.15 is indeed selecting the most *likely* goals amongst the hypothetical goal set $\mathcal{G}_T$, under the assumption of the observed agent being *rational* as in preferring optimal plans over other plans. However, it does not take into account to what extent observing the actions in O might be a *necessary* condition in order to minimize the cost to attain a goal. The following example illustrates this shortcoming.

Figure 4.2 illustrates one case where the cost deviation alone $\Delta(G, O)$ does not account well for the intuitive probability of G given O. In this example, the computation defined at Proposition 4.15 would yield the following $\mathcal{G}_T^*$:

$$\mathcal{G}_T^* = \{G_2, G_5\}$$

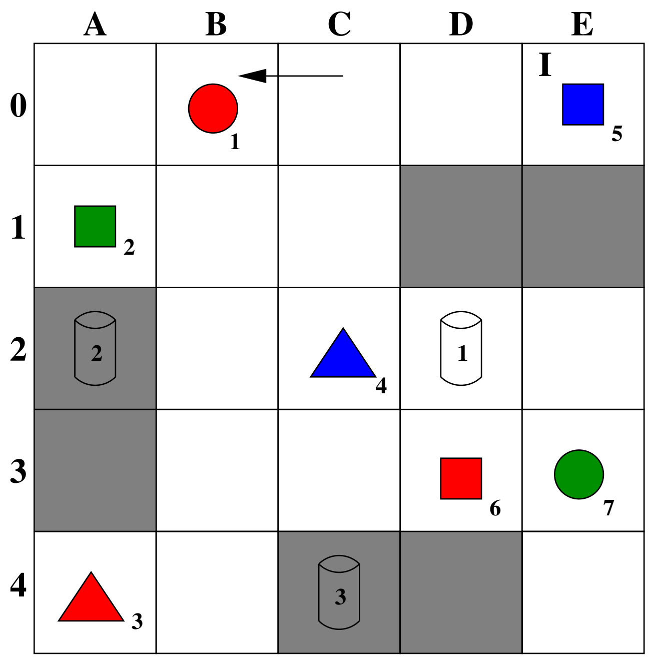

Figure 4.2: RoboSchool plan recognition theory where minimizing $\Delta(G, O)$ does not entail maximizing $P(G|O)$. As in the theory depicted in Figure 4.1, the trainer starts at cell $E0$, but now the observation sequence O is $(\text{walk}(C0, B0))$.

that is, would qualify as equally likely the goals "get all spheres into bin b_2" and "get all the green objects into bin b_3". There are indeed optimal plans satisfying O for both goals, but while for G_2 there is *no* optimal plan that does not satisfy O, for G_5 there are *several* optimal plans not complying with O. Namely, the trainer could make a diagonal move from $C0$ into $B1$, or bypass entirely cell $C0$ going along cells $D0$, $C1$, $B1$ to pick up the green square, without incurring in any additional cost. On the other hand, the minimum cost plans for G_2 that do not satisfy O have a higher cost than those satisfying O. Rather than moving from cell $C0$ into $B0$ with one single walk action, they would involve two, e.g. $\text{walk}(C0, B1)$, $\text{walk}(B1, B0)$ with a cost of $1 + \sqrt{2}$ rather than 1.

In order to do better than this, we need to consider how well does a goal G *predict* observation sequence O. In other words, we want to establish to what degree the satisfaction of O is a *necessary* condition for optimality when pursuing goal G. To leverage this, we can compare $\Delta(G, O)$ with $\Delta(G, \overline{O})$, the deviation from optimal behavior when *not* satisfying O. That is, if we find the latter to be zero, so $c^*_{P'}(G) = c^*_{P'}(G + \overline{O})$, then clearly G is not predicting O very well, since $\Delta(G, \overline{O}) = 0$ tells us that the actions in O are not essential to attain G in an efficient manner.

In the RoboSchool task presented in Figure 4.2, while $\Delta(G, O) = 0$ for both G_2 and G_5, $\Delta(G_2, \overline{O})$ is $2 + \sqrt{2}$, while $\Delta(G_5, \overline{O})$ is 0. So G_2 predicts *better* O under the assumption of observed agents pursuing the most efficient course of action, since plans for $P[G_5]$ which do not satisfy O result in *more* cost efficient plans than those that satisfy O.

Yet another limitation of the algorithm presented in Section 4.5 resides in coping with non–optimal behavior. Figure 4.3 depicts several plan recognition problems over a grid of 11x11 tiles where an agent, initially at the center of the bottom row –

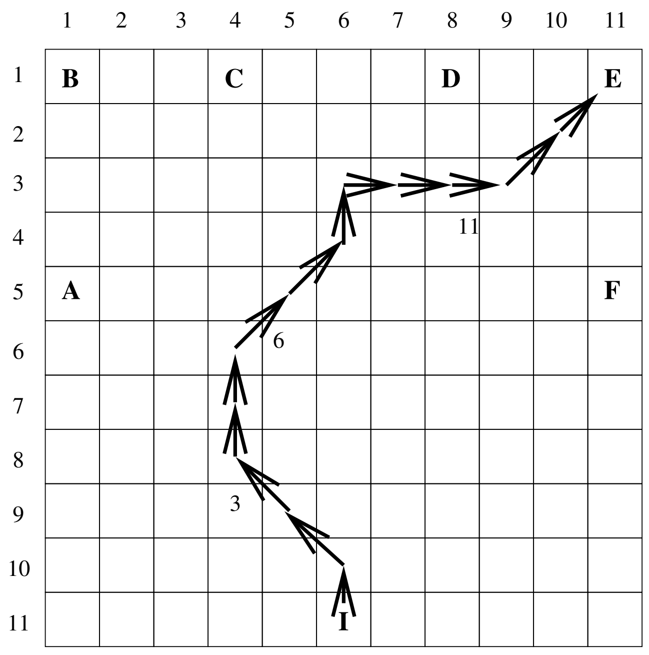

Figure 4.3: Grid navigation problem where observed actions lead to several plausible or no plausible goals.

cell marked I – heads to one of the possible goals A, B, C, D, E or F, by performing three types of moves: horizontal and vertical moves at cost 1, and diagonal moves at cost $\sqrt{2}$. The arrows show the path taken by the agent, with numbers 3, 6 and 11 indicating the time an action was done. Taking as observation sequences O_3, O_6 and O_{11} all actions done by the agent up to time steps 3, 6 and 11 respectively, we obtain three different plan recognition problems T_3, T_6 and T_{11} that only differ in their associated observation sequence O.

As shown in Figure 4.3 the goal being pursued by the agent and eventually achieved is E. We note that this goal would not be considered a plausible one in any of the three plan recognition problems. For T_3 the plausible goal set $\mathcal{G}^*_{T_3}$ would include goals A, B and C, since O_3 is a prefix for all optimal plans achieving either of these three goals. For T_6 and T_{11} $\mathcal{G}^*_T$ would be empty, since observation sequences O_6 and O_{11} are not satisfied by *any* optimal plan for any of the hypothetic goals. Whatever the reason, the observed agent is not pursuing goals in the most efficient way, although it is obvious that for T_6 goal D looks more plausible than goal C, and goal C more plausible than either A or B. When considering T_{11}, goals E and F look to be more plausible than any other goal.

We want a more robust characterization of plausible goal sets, that allows deviations from optimal behavior and allows to *rank* goals according to how likely they are given O. There are several possible settings where this is needed. Namely, in those case where the agent pursuing one of the goals in $\mathcal{G}_T$ *and* another goal which the modeler has not taken into account. Pursuing robustness we will abandon the qualitative notion of $\mathcal{G}^*_T$ altogether and move into a Bayesian probabilistic reasoning setting where goals are ranked according to a posterior probability distribution $P(G|O)$, that describes how well do observation sequences O support the hypothesis of an agent pursuing goal G.

4.7 Probabilistic Model of Plan Recognition

In order to infer the hypothetic goals posterior probability distribution, $P(G|O)$, we need to modify Definition 4.1 to accomodate information about goals *prior* probability distribution.

Definition 4.16. *A probabilistic plan recognition theory or problem is a tuple* $T = \langle P[\cdot], \mathcal{G}_T, O, Prob \rangle$ *where* P *is a planning domain,* $\mathcal{G}_T$ *is a set possible goals* G*,* O *is an* observation sequence *and* $Prob$ *is a probability distribution over* $\mathcal{G}_T$ *.*

Goal priors are supposed to be set accordingly by the plan recognition system designer, to account for knowledge that might make certain goals more likely than others, and which cannot be easily encoded into a planning domain $P[\cdot]$. For instance, in a plan recognition task embedded into a smart house application, the day of the week might and the hour make more or less likely that the observed agent is pursuing the goal "take the car out of the garage".

The posterior goal probabilities $P(G|O)$ will be characterized by the Bayes Rule:

$$P(G|O) = \alpha P(O|G)P(G) \tag{4.1}$$

where α is a normalizing constant and $P(G)$ is $Prob(G)$. The challenge in this formulation is the definition of the likelihoods $P(O|G)$ that express the probability of observing O when the goal pursued is G. Adopting the agent rationality postulate discussed in Section 4.1, that assumes observed agents to pursue goals in the most efficient manner, then $P(O|G)$ has to take into account how much – if at all – does the agent deviate from *optimal* behavior to comply with the observations. The notion of how necessary is O in order to attain the most efficient behavior is implicit in the proportionality between $P(O|G)$ and $P(\overline{O}|G)$. When $P(O|G) = P(\overline{O}|G)$ then clearly O is *not* instrumental in achieving G in the most efficient manner possible.

In the extreme case that a goal G becomes *unfeasible*, that is, there is no valid plan for $P[G]$ that satisfies O, then the likelihood of observing the agent doing the actions in O while pursuing goal G, $P(O|G)$, should be zero. At the other extreme, when there is no valid plan for $P[G]$ that does *not* satisfy O, the likelihood of not observing O when G is being pursued, $P(\overline{O}|G)$, should be zero. When the latter is true, then G is a *perfect* predictor for O.

The natural way to define the likelihoods $P(O|G)$ is

$$P(O|G) = \sum_{\pi} P(O|\pi) \cdot P(\pi|G) \tag{4.2}$$

where π ranges over all possible action sequences, as the observations are independent of the goal G given π. In this expression we can set $P(O|\pi)$ to 1 if π is an action sequence that satisfies O, and 0 otherwise. Moreover, assuming that $P(\pi|G)$ is 0 for π which are not *valid* plans for G, Equation 4.2 can be rewritten as:

$$P(O|G) = \sum_{\pi} P(\pi|G) \tag{4.3}$$

where π ranges now over the valid plans for G that comply with O. Exact computation of $P(O|G)$ as per Equation 4.3, amounts to *count* all plans for $P[G]$. This is something which is likely to be expensive for any planning problem $P[G]$ whose domain $P[\cdot]$ models agent behavior with some degree of flexibility, regarding the ordering of actions in a valid plan or the possibility of attaining certain fluents (sets of fluents) with more than one action.

Approximating $P(O|G)$

We can approximate $P(O|G)$ if we make the following two assumptions:

1. The probability of a plan π for G is proportional to
$$\exp\{-\beta\, c(\pi)\}$$
which follows from our initial assumption of agents pursuing goals in the most efficient manner, and

2. The sum in Equation 4.3 is dominated by its largest term, so that *probabilities corresponding with different plans for the same goal are not added up.*

The first assumption consists in considering probabilities of plans to be inversely proportional to their cost and distributed according to a *exponential* distribution with β being the *rate* parameter. This allows plan recognition system developers to soften the implicit assumption of the agent being rational as in preferring those plans that minimize their total cost. The smaller the value of β the more will the distribution resemble a uniform distribution as probabilities of plans with different costs become more similar.

The second assumption is an order–of–magnitude approximation, of the type that underlies Qualitative Probability Calculus (Goldszmidt and Pearl, 1996). This approximation is reasonable when cheaper plans are much more likely than more expensive plans, and the best plans for $G + O$ and $G + \overline{O}$ are unique or have different costs.

Let us consider the case where the agent has two actions available a and b with cost 1, and the hypothetical goal set $\mathcal{G}_T$ to comprise two goals, G_1 and G_2. Action a achieves G_1 but disables action b by deleting one of the fluents in $Pre(b)$. Action b achieves G_2 but disables action a, since it is deleting a fluent in $Pre(a)$. Now we observe two different performances by the agent, obtaining observation sequences $O_1 = \{a\}$ and $O_2 = \{b\}$.

As we stated above, $P(O|G)$ semantics state how good is G at predicting O. In the case of O_2 and G_1, $P(O_2|G_1)$ should be 0. No rational agent – that is, one seeking goal G_1 – will use action b since that executing that action would indirectly preclude the possibility of achieving G_1 by executing a. On the other hand, the only plan achieving G_1 uses action a but not b, so $P(\bar{O}_2|G_1)$, the likelihood of not having to comply with O_2 to achieve G_1, is 1.

The relationship between likelihoods $P(O|G)$, $P(\overline{O}|G)$ and the cost difference between $c_{P'}(G, O)$ and $c_{P'}(G, \overline{O})$ becomes apparent:

1. When $\Delta(G, O) = \infty$ then $P(O|G) = 0$.
2. When $\Delta(G, \overline{O}) = \infty$ then $P(\overline{O}|G) = 0$.
3. When $\Delta(G, O) < \Delta(G, \overline{O})$ then $P(O|G) > P(\overline{O}|G)$.
4. When $\Delta(G, \overline{O}) < \Delta(G, O)$ then $P(\overline{O}|G) > P(O|G)$.

Likelihoods for $P(O|G)$ and $P(\overline{O}|G)$ follow from our general assumption about non–optimal plans being less likely than optimal plans for G and the assumption made on the probability distribution of $P(\pi|G)$

$$P(O|G) \overset{\text{def}}{=} \alpha' \exp\{-\beta\,(c(G, O) - c(G))\} \tag{4.4}$$

$$P(\overline{O}|G) \overset{\text{def}}{=} \alpha' \exp\{-\beta\,(c(G, \overline{O}) - c(G))\} \tag{4.5}$$

We note that α' is a normalization constant that ensures that $P(O|G)$ and $P(\overline{O}|G)$ add up to 1. If we take the ratio of these two equations, we get

$$P(\overline{O}|G)/P(O|G) = \exp\{-\beta\,\Delta(G)\} \tag{4.6}$$

where $\Delta(G)$ is the cost difference

$$\Delta(G) = c(G, \overline{O}) - c(G, O) \tag{4.7}$$

and the term $c(G)$ appearing in Equations 4.4 and 4.5 is canceled out. Since

$$P(O|G) + P(\overline{O}|G) = 1 \tag{4.8}$$

and given that

$$P(\overline{O}|G) = \exp\{-\beta\,\Delta(G)\}\,P(O|G) \tag{4.9}$$

we can rewrite Equation 4.8 as follows:

$$\begin{aligned} \exp\{-\beta\,\Delta(G)\}\,P(O|G) + P(O|G) &= 1 \\ P(O|G)(1 + \exp\{-\beta\,\Delta(G)\}) &= 1 \\ P(O|G) &= \frac{1}{1 + \exp\{-\beta\,\Delta(G)\}} \end{aligned} \tag{4.10}$$

so $P(O|G)$ corresponds exactly with one member of the class of *sigmoid* functions, a logistic curve with argument $\beta\,\Delta(G)$ as the one shown on Figure 4.4.

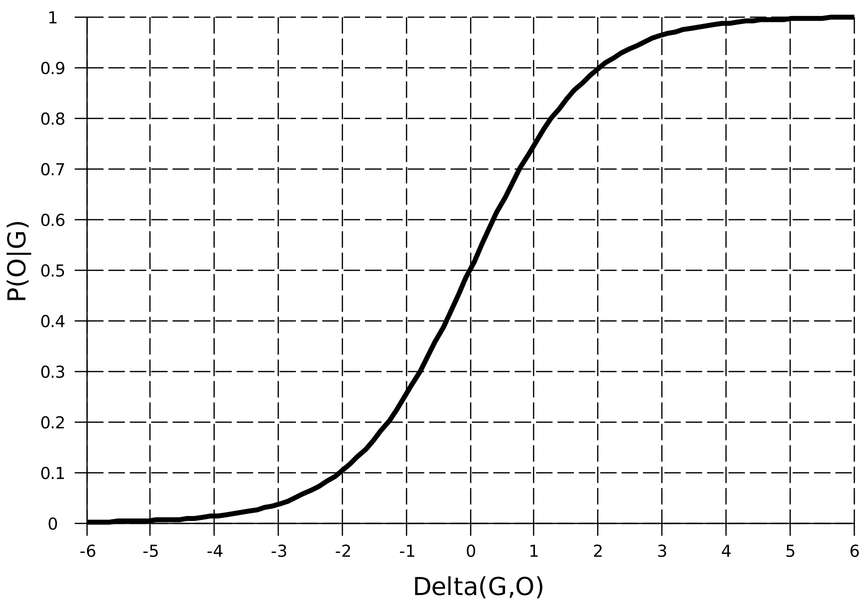

Figure 4.4: $P(O|G)$ as a logistic curve evaluated over $\beta\Delta(G)$ with $\beta = 1$.

4.8 Computation of Posterior Goal Probabilities with Classical Planners

Having characterized in a precise way $P(O|G)$ and given the priors specified in the probabilistic plan recognition problem, one can readily obtain $P(G|O)$ from equation 4.1.

Proposition 4.17 (Computation of $P(G|O)$). *Let $T = \langle P, \mathcal{G}_T, O, Prob\rangle$ be a probabilistic plan recognition problem, and $T' = \langle P', \mathcal{G}_T, O'\rangle$ be the plan recognition problem resulting from compiling observation sequence O into domain P. The posterior distribution $P(G|O)$ for all possible goals $G_i \in \mathcal{G}_T$ is to be computed by:*

1. *Compute $c_{P'}(G_i + O)$ and $c_{P'}(G_i + \overline{O})$ invoking an* optimal *classical planner* twice.
2. *Compute $\Delta(G_i)$ as per Equation 4.7.*
3. *Compute $P(O|G_i)$ as per Equation 4.10.*

This algorithm involves $2|\mathcal{G}_T|$ calls to a classical planner.

When priors are equal, it is trivial to verify that the most likely goals $G \subseteq \mathcal{G}_T$ will be the ones that minimize $\Delta(G, O)$.

Two optimizations can be directly applied on the basic algorithm proposed in Proposition 4.17. The first deals with the overhead entailed by the transformation described in Definition 4.6 and the second with the fact that no plans might exist for $P'[G+\overline{O}]$.

Algorithm 4.5 addresses the first possible optimization by reducing to just one the number of calls over the transformed planning domain $P'[\cdot]$, which might be more expensive than computing a plan for G over the original domain $P[\cdot]$. The algorithm

```
Input: Planning domain $P[\cdot]$
Input: Observation sequence $O$
Input: Hypothetic goal set $\mathcal{G}_T$
Output: Costs $c_{P'}(G+O)$, $c_{P'}(G+\overline{O})$ for each $G \in \mathcal{G}_T$
foreach $G \in \mathcal{G}_T$ do
    Compute optimal plan $\pi$ for $P[G]$ with classical planner;
    if $\pi$ satisfies $O$ then
        Set $c_{P'}(G+O) = c(\pi)$;
        Compute $c_{P'}(G+\overline{O})$ with classical planner;
    else
        Set $c_{P'}(G+\overline{O}) = c(\pi)$;
        Compute $c_{P'}(G+O)$ with classical planner;
```

Figure 4.5: Optimized computation of $c_{P'}(G+O)$ and $c_{P'}(G+\overline{O})$

exploits the result in Theorem 4.11 that states that plans over domain $P[\cdot]$ and $P'[\cdot]$ for the same goal G are equivalent.

For the second optimization we will rely on the notion of *action landmarks* discussed in Section 2.8, actions that are *necessarily* featured by all valid plans for some goal G. In certain planning domains, goals $G + \overline{O}$ may become unreachable from the initial state I' when one of the actions in O happens to be an action landmark for either the goal G or a precondition of another observed action:

Proposition 4.18. *Given that $c(G) \neq \infty$ and observation sequence $O = (a_1, \dots, a_n)$, $c_{P'}(G+\overline{O}) = \infty$ iff:*

1. *$a_n \in O$ last action in O, a_n action landmark for G, and*
2. *Observation a_i is an action landmark for $Pre(a_{i+1})$*

Action landmarks can be computed inexpensively with the h^1 max heuristic described in Section 2.6 as shown on Algorithm 4.6.

```
Input: Planning domain $P[\cdot] = \langle F, I, A \rangle$
Input: Observation sequence $O$
Input: Goal $G$
Remove $a_n$ from $A$, yielding planning domain $P_{a_n}[\cdot]$;
Compute $h_{max}(G; I)$ over domain $P_{a_n}[\cdot]$, if $\infty$ return $False$;
foreach $a_i \in O$, $i = 1, \dots, n-1$ do
    Remove $a_i$ from $A$, yielding planning domain $P_{a_i}[\cdot]$;
    Compute $h_{max}(Pre(a_{i+1}); I)$ over domain $P_{a_i}[\cdot]$, if $\infty$ return $False$;
Return $True$;
```

Figure 4.6: Algorithm for checking that $P'[G+\overline{O}]$ is solvable.

This relative inexpensive computation allows huge savings when $P'[G + \overline{O}]$ is not reachable from the initial state I', since planning heuristics other than the h^m heuristics with $m > 1$ cannot detect *dead ends*.

Example

Figure 4.7 illustrates how the posterior distribution $P(G|O)$ changes as we observe more of the agent behavior shown in Figure 4.3. At each time step t, observation sequences O_t contain *all* the actions done by the agent up to t.

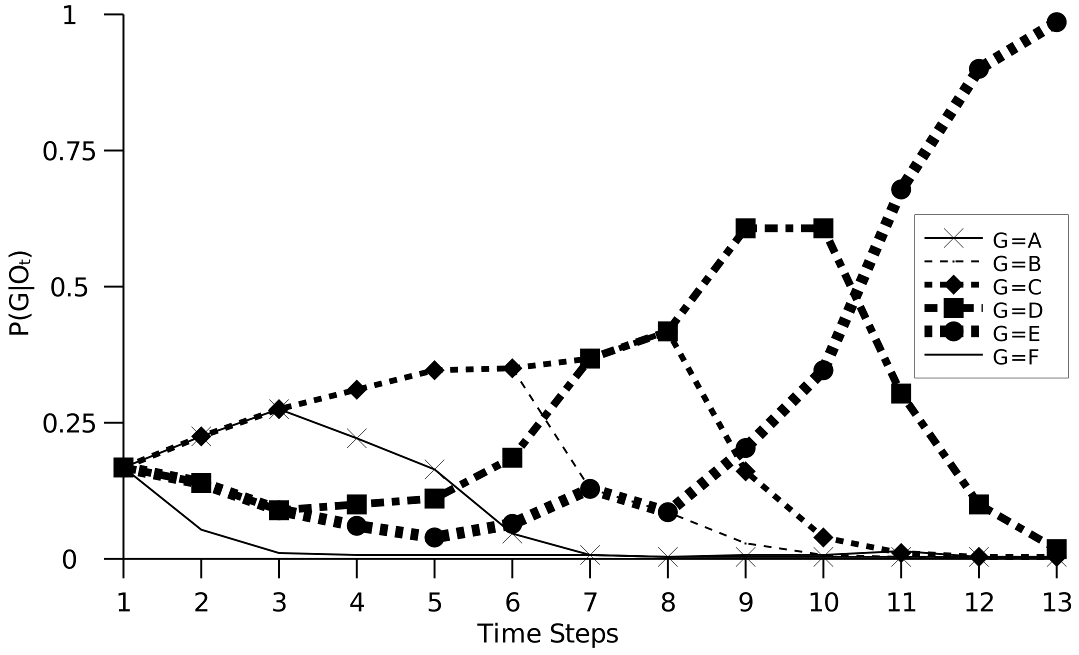

Figure 4.7: $P(G|O_t)$ as function of time t.

At each time step t $P(G|O_t)$ is computed from scratch using the algorithm described in Proposition 4.17 while updating observation sequence O_t. Note that computing $P(G|O_t)$ will involve solving optimally 26 classical planning problems, 2 for each of the 13 time steps.

As it can be seen on Figure 4.7, until time step 3 the most likely goals are A, B and C, which corresponds exactly with our intuition. Between time steps 3 and 7 we see how first goal A and then goal B likelihoods sharply decrease as the agent departs from the optimal plans to achieve them, being replaced by goal D. From time step 7 onwards we see how goal E likelihood raises steadily as the agent progresses towards the top–right corner of the grid, while the other goals likelihoods become nearly zero as the agent reaches goal E. It is interesting to note that the likelihood of goal F never ever really got off the ground, even at time step 11 after the agent goes towards the right edge of the grid. At time step 11 the cost to reach F and comply with O_{11}, $c_{P'}(F + O_{11})$, is $6 + 6\sqrt{2} \approx 14.5$ nearly twice as much as the cost of reaching F without complying with O_{11}, $c_{P'}(F + \bar{O_{11}})$, which is $1 + 5\sqrt{2} \approx 8$.

Using Satisficing Classical Planners to Approximate $P(G|O)$

While Proposition 4.17 uses optimal classical planners to compute costs $c_{P'}(G+O)$ and $c_{P'}(G+\overline{O})$, in practice, optimal classical planners do not scale up as well as satisficing classical planners do. In the last decade the classical planning community has devoted huge efforts to develop very efficient satisficing planners as discussed in Chapter 2. Satisficing planners *do* scale well, so practical plan recognition systems built on top of the principles presented in this thesis can use them as the core of their processing.

Nonetheless, there is one important *caveat* about using satisficing planners. Costs computed by satisficing planners are *not guaranteed to be a lower–bound on plan cost.* Empirical results for state–of–the–art satisficing planners show that, depending on the structure of the planning problem considered, there is a *huge* variability on how close reported solutions costs are from optimal solutions costs. This can potentially lead to very different posteriors so a goal G that maximizes $P(G|O)$ when computed with an optimal planner, might not do so when $P(G|O)$ is computed with a satisficing planner.

Anytime planning systems such as LAMA (Richter and Westphal, 2008), allow us to control to what side in the trade–off between scalability and robustness inferences fall by allocating a bounded amount of time to plan search. We will further analyze this issue when we present the empirical evaluation of our approach to probabilistic plan recognition in Section 4.11.

4.9 Evaluation Domains from Planning Benchmarks

In this section we will present three different plan recognition tasks which we will use to empirically test the formulation we have presented in Section 4.7. For each domain we will describe the associated planning domains and hypothetical goal sets $\mathcal{G}_T$, so the reader can appraise the expressiveness contributed by using a relatively simple modeling language such as STRIPS and compare it with previous approaches to plan recognition.

Block Words

In this plan recognition task, the agent wants to assemble a word from an English word vocabulary by assembling a tower of toy blocks with letters painted on them. The observer expects the agent to aim at one of 20 possible words made up by up to 6 different letters.

The action theory modeling agent behavior is obtained by borrowing the BLOCK WORLD domain introduced in the 2nd International Planning Competition benchmarks [4]. In this planning domain the agent can interact with several blocks located on top of a table with an unlimited surface. Blocks can initially be either on top of the table or on top of another block.

[4]Available at `http://www.loria.fr/~hoffmanj/ff-domains.html` under the label `Blocksworld-4ops` (retrieved on October 2011).

Fluent	Meaning
clear(b_i)	Block b_i has no other block b_j on top.
on(b_i, b_j)	Block b_i is on top of block b_j.
onTable(b_i)	Block b_i lays directly on the table.
holding(b_i)	Agent holds block b_i in her hand.
handEmpty()	Agent hand is empty.

Table 4.2: BLOCK WORLDS fluent set F.

Fluents describing world states are listed on Table 4.2. The agent can manipulate the blocks in several ways to arrange them into one or more towers. Agent actions STRIPS description is listed on Table 4.2.

Action	Preconditions	Adds	Deletes

pickUp(b_i)	clear(b_i)
	onTable(b_i)
		holding(b_i)	clear(b_i)
			handEmpty()

putDown(b_i)	holding(b_i)	handEmpty()
		onTable(b_i)
			holding(b_i)

stack(b_i, b_j)	holding(b_i)
	clear(b_j)
		on(b_i, b_j)
		handEmpty()
		clear(b_i)
			clear(b_j)
			holding(b_i)

unstack(b_i, b_j)	on(b_i, b_j)
	clear(b_i)
	handEmpty()
		holding(b_i)
		clear(b_j)
			on(b_i, b_j)
			clear(b_i)
			handEmpty()

Table 4.3: BLOCK WORLDS STRIPS action set A.

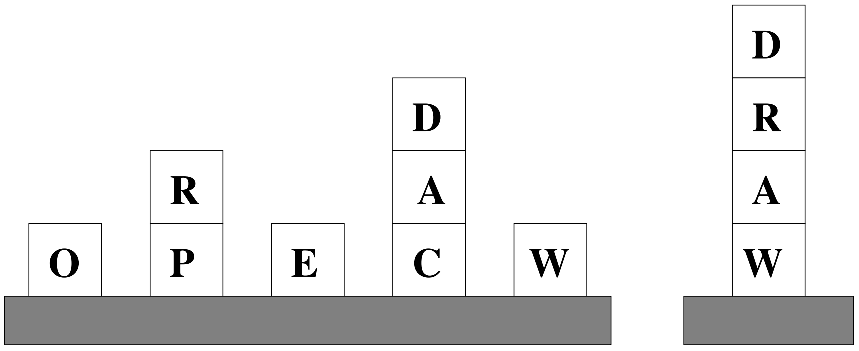

Figure 4.8: BLOCKWORDS PR task example. On the left is shown the initial configuration I, and on the right one of the possible goals $G \in \mathcal{G}_T$.

Initial states I can describe any valid configuration of blocks such as the one depicted on Figure 4.8. The set of possible goals $\mathcal{G}_T$ is encoded as *conjunctions* of fluents shown on Table 4.2 describing towers of blocks, whose letters when read from top to bottom, configure some English word. Figure 4.8 shows the tower of blocks corresponding with the word "draw". In our experiments we set $\mathcal{G}_T$ to include twenty different towers corresponding with English words.

One of the many [5] optimal plans for the planning problem $P[G]$ in Figure 4.8 could be:

1. unstack(D, A)	2. putDown(D)	3. unstack(A, C)
4. stack(A, W)	5. unstack(R, P)	6. stack(R, A)
7. pickUp(D)	8. stack(D, R)	

We note that there is potentially a high degree of ambiguity between different goals, since it is very likely that partially observed plans for different goals will have in common a substantial number of actions, and these tend to as well ordered in the same way. Consider the plan for the goal consisting on building a tower reading the word "crow":

1. unstack(D, A)	2. putDown(D)	3. unstack(A, C)
4. putdown(A)	5. pickUp(O)	6. stack(O, W)
7. unstack(R, P)	8. stack(R, O)	9. pickUp(C)
10. stack(C, R)		

If the observed sequence of actions is $O = \{$ putDown(D), unstack(R, P) $\}$ both goals, that is assembling towers for words "draw" and "crow", will have the same probability $P(G|O)$.

Block Words is a challenging domain for approaches to plan recognition based on plan libraries because of the very high number of valid plans existing even for simple tasks as the one depicted in Figure 4.8. All action effects are *reversible* so there several optimal goals and a huge number of non–optimal valid plans. In our formulation, the huge search space entailed by this domain poses a challenge to optimal planners. On the other hand, satisficing planners find plans very quickly, though sometimes these plans quality is substantially worse than that of optimal plans.

Logistics

Here the task for the observer is to deduce the destination – locations in two cities – of up to 6 different packages. Packages start in locations in the same or a different city than that of their destination, and can be located at either locations or airports, or inside trucks and airplanes. Table 4.4 details the fluents F in the planning domain $P[\cdot]$ of the plan recognition problem.

In order to deliver the packages, trucks can load and unload them and travel between locations in the same city. When a package needs to be delivered to a location in a different city, it is required to ferry it by airplane, hauling it first to the city airport. Table 4.5 detail actions in the planning domain $P[\cdot]$ associated with these plan recognition problems.

In our experiments we have considered five different possible initial situations I, by assigning randomly trucks, packages and airplanes to suitable locations. The

[5]All other optimal plans for the goal shown in Figure 4.8 involve stacking block D on top of any block other than W.

Fluent	Meaning
inCity(l, c)	Location l is in city c.
at(o, l)	Object o, either a truck or a package, can be found at place l, either a location or an airport.
in(p, v)	Package p is located inside vehicle v, either a truck or an airplane.

Table 4.4: LOGISTICS fluent set F.

Action	Preconditions	Adds	Deletes
loadTruck(p, t, l)	at(t, l) at(p, l)	in(p, t)	at(p, l)
loadAirplane(p, a, l)	at(a, l) at(p, l)	in(p, a)	at(p, l)
unloadTruck(p, t, l)	in(p, t) at(t, l)	at(p, l)	in(p, t)
unloadAirplane(p, a, l)	in(p, a) at(a, l)	at(p, l)	in(p, a)
driveTruck(t, l_1, l_2, c)	$l_1 \neq l_2$ at(t, l_1) inCity(l_1, c) inCity(l_2, c)	at(t, l_2)	at(t, l_1)
flyAirplane(a, l_1, l_2, c)	$l_1 \neq l_2$ at(a, l_1)	at(a, l_2)	at(a, l_1)

Table 4.5: LOGISTICS STRIPS action set A. p are packages, t trucks, l locations

hypothetical goal set $\mathcal{G}_T$ consists of up to 10 different combinations of assignments of packages to locations such as

$$\begin{gathered}
G_1 : \mathrm{at}(p_{1_1}, l_{2_1}), \mathrm{at}(p_{2_3}, l_{1_3}) \\
G_2 : \mathrm{at}(p_{1_2}, l_{2_3}), \mathrm{at}(p_{2_2}, l_{1_3}) \\
G_3 : \mathrm{at}(p_{1_3}, l_{2_1}), \mathrm{at}(p_{2_2}, l_{1_1}) \\
G_4 : \mathrm{at}(p_{1_3}, l_{2_2}), \mathrm{at}(p_{2_3}, l_{1_3}) \\
\ldots\ldots
\end{gathered}$$

where goal G_1 would read aloud as "package p_{1_1} is at location l_{2_1} *and* package p_{2_3} is at location l_{1_3}".

The *Logistics* domain is not a challenge for the state–of–the–art in either optimal or satisficing planning. On the other hand, it is simple but expressive enough to capture an interesting (Liao et al., 2007) plan recognition task, where the degree of precision we can achieve identifying the actual goal being pursued is clearly related to the number of fluents goals G_i do share.

IPC–Grid

The planning domain for this collection of plan recognition tasks is inspired on 2nd International Planning Competition GRID benchmarks [6]. The observed agent navigates through a graph, where nodes model rooms and edges between nodes denote that there is a door between two rooms. Figure 4.9 depicts one of the graphs we consider.

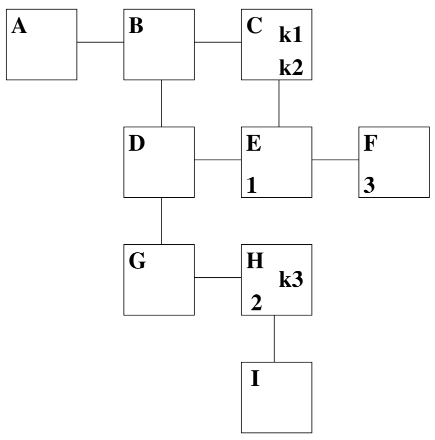

Figure 4.9: GRID PR task example. Boxes are rooms and lines between them denote that they are connected. Lock shapes – numbers 1, 2 and 3 – are shown only for locked rooms E, F and H. k_1, k_2 and k_3 are keys whose shapes correspond with that of locks of rooms E, H and F respectively. The observed agent starts in room A.

These rooms might be either open or closed. In the latter case the agent cannot enter the room, unless she has the right key to open it. Keys can open all rooms whose lock shape matches that of the key. In the tasks included in our benchmark there are up to five possible key shapes and types of locks. Tables 4.6 and 4.7 show the fluent and action set of the planning domain.

Fluent	Meaning
connected(x, y)	Rooms x and y are connected by a door
keyShape(k, s)	Key k has shape s
lockShape(x, s)	Lock at room x has shape s
atKey(k, x)	Key k is in room x
atAgent(x)	Agent is in room x
locked(x)	Room x is locked
carrying(k)	Agent is carrying key k
open(x)	Room x is open

Table 4.6: GRID fluent set F.

Agent goals are to get into one room, which correspond with goals consisting of one single atAgent(x) fluent. Since connections between rooms are quite sparse and several keys might be placed in the same room, goal rooms widely separated on the graph might predict equally well many observation sequences.

[6]Available at `http://www.loria.fr/~hoffmanj/ff-domains.html` under the label `Grid` (retrieved on October 2011).

Action	Preconditions	Adds	Deletes
unlock(x, y, k, s)	connected(x, y) keyShape(k, s) lockShape(x, s) atAgent(x) locked(y) carrying(k)	open(x)	locked(x)
move(x, y)	atAgent(x) connected(x, y) open(x)	atAgent(x)	atAgent(y)
pickUp(x, k)	atAgent(x) atKey(k, x)	carrying(k)	atKey(k, x)

Table 4.7: Grid Strips action set A.

As an illustration of the observation above, let us consider the following two goals for the task depicted in Figure 4.9:

$$G_1 : \text{atAgent}(F)$$
$$G_2 : \text{atAgent}(I)$$

Plans for both G_1 and G_2 involve moving to room C to pick up just k_2, when the actual goal is G_2, or both k_1 and k_2, when the actual goal is G_1. Note k_2 is also relevant for G_1, since it allows to unlock room H and pick up k_3 which unlocks room F.

4.10 Evaluation Domains from Plan Libraries

This section discusses three benchmarks obtained from plan recognition tasks discussed in recent literature on plan recognition (Geib and Goldman, 2001; Wu et al., 2007; Bui et al., 2008). This presented a challenge because of the general vagueness when defining the modeling language used to describe plan recognition tasks. In general, the plan recognition community has settled for a simplified form of the Htn modeling language (Erol et al., 1994) to specify in a compact way plan libraries.

Compared with Strips, Htn not only provides with the description of world states and what actions the agent can use to affect these, but also with strategies for executing *tasks*. Htn tasks are the rough equivalent of Strips *goals*, and Htn domain definitions can indeed be understood as a library of plans, as *methods* are actually short plans prescribing how to achieve some simpler task or goal.

We settled for establishing a compromise between a faithful rendition of the original problem semantics – giving formal guarantees about correspondence of solutions when possible – and showcasing the flexibility of Strips when it comes to model plan recognition tasks. This flexibility allows us to transform plan libraries into Strips action theories whose valid plans correspond with the plans in the library, as well as taking into account additional constraints entailed by the context where the observed agent performance takes place.

While algorithms exist to compile HTN decompositions into STRIPS planning domains (Lekavý and Návrat, 2007), we chose to translate the ones on Figure 4.10 with the following mapping into STRIPS:

Definition 4.19. *Let $L = \langle \mathcal{T}, T, Sub, Depend \rangle$ be a hierarchical plan library, where $\mathcal{T}$ are top–level tasks, T are primitive tasks t and "end tasks" End_τ, Sub are tuples (τ, t), $\tau \in \mathcal{T}$, $t \in T$, denoting that t is a* subtask *of τ and $Depend$ are tuples (t, t') denoting that $t' \in T$ requires $t \in T$ to be completed before being executed. The planning domain $P_L[\cdot] = \langle F, I, A \rangle$ resulting from compiling L into* STRIPS *is defined as follows:*

1. $F = \{done(t) \,|\, t \in T\} \cup \{completed(\tau) \,|\, \tau \in \mathcal{T}\}$
2. $I = \emptyset$
3. $A = A^T \cup A^{\mathcal{T}}$ *where* $A^T = \{a_t \,|\, t \in T\}$, *$t$ not an end action*
   a) $Pre(a_t) = \{done(t') \,|\, (t', t) \in Depend\}$
   b) $Add(a_t) = \{done(t)\}$

   and $A^{\mathcal{T}} = \{a^\tau_{end} \,|\, \tau \in \mathcal{T}\}$ *where*
   a) $Pre(a^\tau_{end}) = \{done(t) \,|\, (t', End_\tau) \in Depend\}$
   b) $Add(a^\tau_{end}) = \{completed(\tau)\}$

A decomposition δ is a *path* – sequence of edges – that traverses the graph implicitly defined by the library L, subject to some constraints:

Definition 4.20 (Library graph $G(L)$)**.** *The* library graph *$G(L) = (V, E)$ vertex set V is defined as the union of the library top–level and primitive tasks*

$$V = \mathcal{T} \cup T$$

and edge set E

$$E = \{(\tau, t), (t, \tau) \,|\, (\tau, t) \in Sub\} \cup Depend$$

Definition 4.21 (Decomposition of top–level task τ)**.** *The decomposition of a top–level task $\tau \in \mathcal{T}$ of library L is a* path *δ*

$$\delta = (e_1, e_2, \ldots e_n)$$

such that e_i belongs to the edge set E of $G(L)$, and satisfying the following conditions:

- $e_1 = (\tau, t)$, $(\tau, t) \in Sub(\tau)$
- $e_n = (t', \tau)$, $(\tau, t') \in Sub(\tau)$
- *For all* $t \in T$, *such that* $t \in e_i$, $Depend(t) \subset \delta$

where $Sub(\tau) = \{(\tau, t) | (\tau, t) \in Sub\}$ *and* $Depend(t) = \{(t', t) \,|\, (t', t) \in Depend\}$.

A decomposition δ is *minimal* when

$$|\delta| = \min_{\delta' \in \Delta(L,\tau)} |\delta'|$$

that is, consists of the minimum possible number of steps amongst all possible decompositions of top–level task τ in the library L, $\Delta(L, \tau)$.

The relation between minimal decompositions δ of top–level tasks τ in L and optimal plans in P_L for goals $G = \{completed(\tau)\}$ is described in the following theorem:

Theorem 4.22. *There exists an exact correspondence between* minimal *decompositions for top–level tasks τ in L and valid, optimal plans for $P_L[G]$, where $G = \{completed(\tau)\}$ and $c(a) = 1$ for all actions a.*

Proof. We prove the theorem above by contradiction. Let $\pi = \{a_1, \ldots, a_i, \ldots, a_j \ldots, a_n\}$ be a valid, optimal plan for $P_L[G]$ which does not correspond to a minimal decomposition for τ in L and s_i be the set of fluents $p \in F$ true after $a_1, \ldots, a_i$ is executed. Since π is a valid plan then necessarily $completed(\tau) \in s_n$. Because of π being optimal, the shortest possible valid plan, we can assume that action $a_n = a^{\tau}_{end}$. Again, since π is optimal, it must hold for any set of fluents s_i that $Pre(a_{i+1}) \subseteq s_i$. We observe that all preconditions are composed of $done(t)$ fluents, which are only added by the corresponding a_t action. Then, there must exist actions a_i, a_j in π such that $a_i = a_j = a_t$. In that case, π would not be an optimal plan, since there are no actions in $P_L[G]$ deleting fluents $done(t)$ and hence a_t actions need to be executed *exactly once.* $\square$

Theorem 4.22 establishes a link between past approaches to plan recognition using plan libraries and the formulation given in Section 4.7. Under the assumption of rationality embodied in the optimality of plans, the most likely observation sequences will be those that satisfy some *optimal* plan in P_L, and those plans correspond with the minimal decompositions of plan libraries L as shows Theorem 4.22.

Intrusion Detection

This task is adapted from the application of plan recognition on intrusion detection systems (IDS), an interesting and relevant compute security domain, described in (Geib and Goldman, 2001). In this case the observing agent is a surveillance system which gets to observe the interaction of a malicious computer hacker with a network server. Observed agent goals and plans are encoded into simple hierarchical plans (Erol et al., 1994) as depicted on Figure 4.10.

We have made one change to (Geib and Goldman, 2001) which consists in modeling the routines above so plan recognition tasks could accomodate multiple, concurrent attacks on several different servers. This amounts to add a new type of object – *server* – and modify accordingly predicate and action schemas. This simple change allows for a more realistic scenario and also provides us with a parameter that allows us studying how well our approach scales.

In order to generate the plan recognition tasks used in the experimental evaluation, we take care of excluding from generated observation sequences O actions a^{τ}_{end}, which would indeed give the game away.

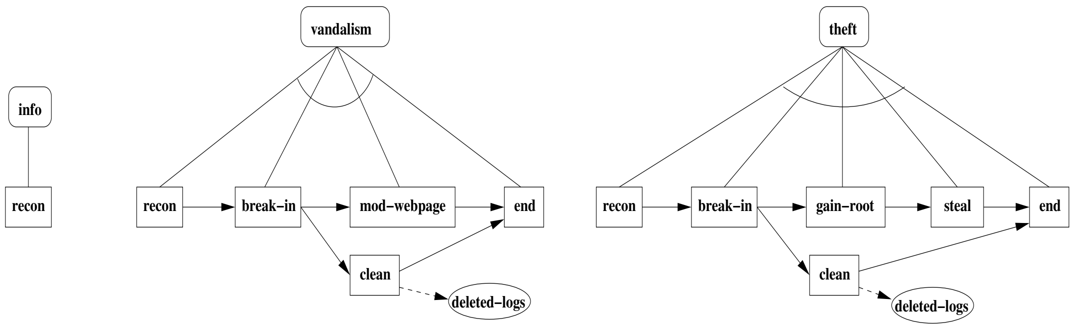

Figure 4.10: Hierarchical plan libraries for the Intrusion Detection task (Geib and Goldman, 2001). Square boxes denote primitive tasks and rounded boxes denote composite tasks and ellipses denote primitive tasks "side effects". Solid arrows between primitive tasks are precedence constraints and discontinue arrows relate primitive tasks and effects.

Campus

In Campus plan recognition tasks the observer agent needs to find out the activity being performed by a college student, whose movements around the campus are being tracked (Bui et al., 2008). Possible plan libraries, rather than being full–fledged hierarchical plans are specified as directed acyclic graphs of primitive tasks, which (Bui et al., 2008) refer to as *routines*. An example of one of the two routines discussed in (Bui et al., 2008) is shown on Figure 4.11.

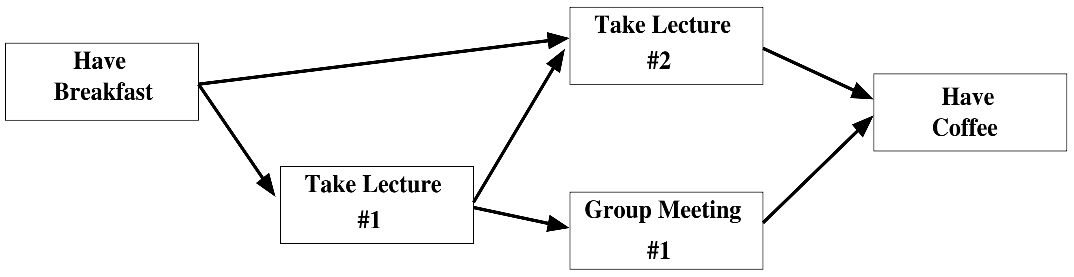

Figure 4.11: Student routine from (Bui et al., 2008) which consists of having breakfast, attending to classes, meeting with other students, and ending with having coffee.

We note that these activities are hidden from the observer, all the info she gathers concern student changes in location. Such changes are providing indirect evidence for a certain activity being executed, since they can only take place in certain campus areas.

We have encoded this task into a Strips action theory by, first, encoding the task dependency graph as described on Definition 4.19, and second, modeling the observed changes in location as Strips actions that allow the student agent to navigate across a fully connected graph, whose vertices correspond with campus landmarks. Fluents and actions in our resulting model are depicted on Table 4.8 and 4.9.

For our experiments we have considered five different *landmarks*. Four are meant to be surrogates for campus buildings, two buildings for the classrooms where lectures take place, another for the campus library and a fourth one for the students' cafeteria. The fifth is representing the entrance to the campus, and is the location where the student starts in our benchmarks. Adding a more complex and constrained

Fluent	Meaning
done(t)	Task t has been executed
at(l)	Student at landmark l
canDo(t, l)	Task t can be performed at landmark l

Table 4.8: CAMPUS fluent set F.

navigation map would just require to add more landmarks to the underlying navigation map and introduce *connected*(x, y) fluents to denote the possibility of moving directly from landmark x to landmark y.

Action	Preconditions	Adds	Deletes
move(x, y)	at(x)	at(y)	at(x)

do(t, l)	canDo(t, l)
	done(t'), $\forall (t', t) \in Depends$
		done(t)	

Table 4.9: CAMPUS STRIPS action set A.

The hypothetical goal set $\mathcal{G}_T$ for each routine consists on having performed all of the tasks in its corresponding activity dependency graph. The goal corresponding with the graph in Figure 4.11 would be the following conjunction of fluents from Table 4.8:

$$\begin{aligned} G: \quad & \text{done(haveBreakfast), done(takeLecture1),} \\ & \text{done(takeLecture2), done(groupMeeting1),} \\ & \text{done(haveCoffee)} \end{aligned}$$

We note that STRIPS semantics enforce that the tasks have been executed in the proper order.

This two–level modeling allows us to pose interesting subtle problems to our plan recognition scheme. One possible observation sequence O could be:

$$move(entrance, cafeteria)$$

This observation would make routines involving the tasks *haveBreakfast* and *haveCoffee* to be more likely than those not featuring them. Another interesting sequence would be

$$O = (move(entrance, cafeteria), move(library, cafeteria))$$

in this case, routines (goals) involving performing an activity in the cafeteria before going to the library to do something else and engaging in another task in the cafeteria *afterwards* would be more likely than routines not featuring this basic structure.

Kitchen

KITCHEN plan recognition tasks are taken from (Wu et al., 2007). The authors discuss an application of activity recognition to the problem of *home automation* or *domotics*. The observing agent is meant to be such a system, which needs to recognize the tasks being done by people dwelling in the house, in order to assist them. For instance, observing one house dweller to take a milk carton from the fridge - possible by rigging the carton with a RFID emitter - could lead the system to conclude that the person is going to prepare breakfast and proceed by starting the induction cookers or programming the microwave oven.

As in (Bui et al., 2008) the authors provide an activity dependency graph and make a distinction between observable activities – taking objects from containers and using appliances – and non–observable activities – such as "make a toast". Figure 4.12 depicts the activity dependency graph for the "make breakfast" activity.

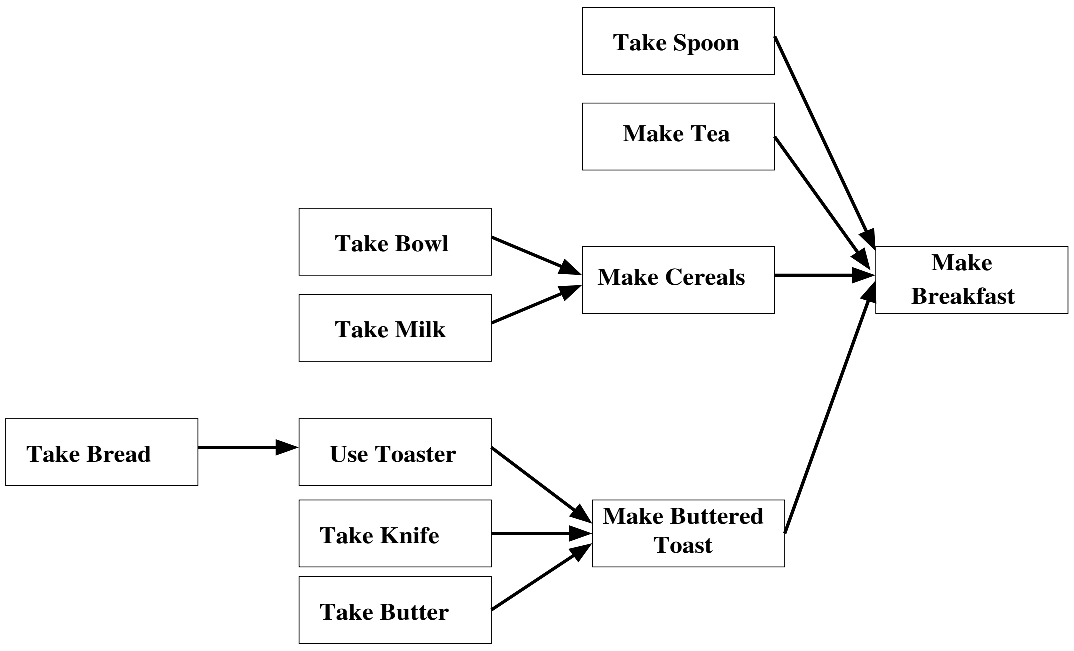

Figure 4.12: MAKE BREAKFAST activity (Wu et al., 2007).

We obtain STRIPS planning domain – fluent and action sets are depicted on Tables 4.10 and 4.11 – applying the translation procedure described in Definition 4.19 only to non–observable activities. Observable activities are modeled as "natural" STRIPS fluents and actions, and non–observable activities requiring one or more observable activities have in their preconditions fluents representing statements such as "the agent holds object o" or "appliance a is on".

Fluent	Meaning
inside(o, c)	Object o is inside container c
done(t)	Task t has been executed
holding(o)	Agent is holding object o
turnedOn(a)	Appliance o is turned on

Table 4.10: KITCHEN fluent set F.

All objects o, that are representing a variety of foodstuffs, start inside some container c, such as the fridge, or one of the kitchen drawers. Observation sequences are limited to contain either $take(o, c)$ or $turnOn(a)$ actions.

Action	Preconditions	Adds	Deletes
take(o, c)	inside(o, c)	holding(o)	inside(o, c)
turnOn(a)		turnedOn(a)	
do(t)	$holding(o)$ for o relevant $turnedOn(a)$ for a relevant done(t'), $\forall\, (t', t) \in Depends$	done(t)	

Table 4.11: Campus Strips action set A.

"Top–level" or rather, final, activities in (Wu et al., 2007) might be completed by different routines. This can be easily handled in a way analogous as how *disjunctive preconditions* are handled when instantiating Pddl schemas into Strips. We allow for this by introducing as many Strips actions adding $achieved(t)$ fluents as optional routines are specified, each action having a precondition that involves having performed the prescribed activities.

4.11 Experimental Results

We have tested the formulation given for probabilistic plan recognition over action theories described in Section 4.7 on six different benchmarks, which have already been presented in Section 4.9. We will next describe how the plan recognition theories are generated, give some implementation notes on the plan recognition algorithm and finish with a discussion of the results of our evaluation. All the problems and code used in the evaluation can be found at `https://sites.google.com/site/prasplanning`.

Implementation Notes

Plan recognition theories $\mathcal{T} = \langle P, O, \mathcal{G}_T, Prob \rangle$ elements are represented in the following way:

- $P[\cdot]$ is given as a template of a planning task definition in Pddl. In particular, hooks for injecting the appropriate Pddl statements are provided in action effects, initial and goal situations. This allows to easily compile observation sequences O automatically and simplifies the implementation of the transformation described in Definition 4.6.

- O is given as a list of names of Strips actions, that is, Pddl action schemas with variables already substituted by suitable objects and constants.

- $\mathcal{G}_T$ is a list of conjunctions of Strips propositional fluents, that is, Pddl predicate schemas with variables substituted by the appropriate object and constant names.

- $Probs$ is a list of real numbers, where each element in the list corresponds with one element in $\mathcal{G}_T$ and denote each hypothetical goal prior probability.

The input above is then compiled into propositional STRIPS, generating for each of the hypothetic goals $G \in \mathcal{G}_T$ two planning problems $P[G+O]$ and $P[G+\overline{O}]$. These two problems are fed into a classical planner and their output – reporting failure to find a solution or a plan and its cost – is processed to obtain $P(G|O)$. We did not modify the code of the planners in any case. The planners source code was compiled and the binary executable was invoked over the UNIX shell. The code driving the whole process was written in PYTHON.

The planners we have used in our evaluation are two. HSP$^*_F$ is an optimal planner (Haslum et al., 2005; Haslum, 2008) that uses the admissible h^m heuristic h^2 described on Section 2.6. The other planner we have used is LAMA (Richter et al., 2008), a satisficing planner discussed in Section 2.9. LAMA is used in its two modes operation available: as a *greedy* planner that stops after the first plan is found, and as an *anytime* planner that reports the best plan found in a given time window. The running times for HSP$^*_F$, anytime LAMA, and greedy LAMA were limited to four hours, 240 seconds, and 120 seconds respectively, *per plan recognition problem.* Each plan recognition problem involves $|\mathcal{G}_T|$ possible goals, requires the computation of $2|\mathcal{G}_T|$ costs, and hence, involves running the planners over $2|\mathcal{G}_T|$ *planning problems.* Thus, on average, each of the planners needs to solve this number of planning problems in the given time window. The experiments were conducted on a dual-processor Xeon 'Woodcrest' running at 2.33 GHz and 8 Gb of RAM. All action costs have been set to 1.

Generation of Observation Sequences

We generated a total of 90 different plan recognition problems, applying the following process to each of the domains considered:

1. A goal $G \in \mathcal{G}_T$ is chosen with uniform probability.
2. A plan $\pi = (a_1, \ldots, a_n)$ is computed for the planning problem $P[G]$.
3. A first observation sequence is obtained from taking π on an as is basis.
4. Four observation sequences are obtained from π by sampling 10%, 30%, 50% and 70% of action indices $i \in [1, n]$ uniformly without replacement. The sampled indices are then ordered to yield observation sequence O.

For the domains BLOCKSWORDS, LOGISTICS, IPC–GRID and INTRUSION DETECTION, plans π are obtained from invoking the optimal classical planner HSP^*_f on $P[G]$. Plans for CAMPUS and KITCHEN were obtained choosing at random one of the plans encoded by the plan library for goal G, and embedding into them subsequences of randomly chosen plans for other goals G' in the library. The length of these subsequences varies randomly between 10% and 70% of the actions in the plan for G'.

Results

The results over the six domains are summarized in Table 4.12. The number of observations in each row correspond with the percentage of actions sampled from the plan π for the hidden goal G: 10%, 30%, 50%, 70%, and 100% as shown. For each domain, the average size of the set $\mathcal{G}_T$ is shown.

		HSP^*_f				LAMA (240s)			Greedy LAMA		
Domain	O	T	Q	S	L	T	Q	S	T	Q	S
	10	1184.23	1	6	10	228.04	0.75	4.75	52.79	0	1.67
Block	30	1269.31	1	3.25	11	239.59	1	3	53.01	0.5	2
Words	50	1423.05	1	2.23	11	241.77	1	2.23	53	0.54	1.23
$\lvert\mathcal{G}\rvert = 20$	70	1787.67	1	1.27	12	241.53	1	1.27	53.06	0.73	1.2
	100	2100.21	1	1.13	12	241.51	1	1.13	53.47	0.73	1.07
	10	73.38	0.75	1.38	15	22.15	0.75	1.38	3.96	0.75	1.38
Easy	30	155.47	1	1	17	64.63	1	1	5.38	1	1.08
Ipc	50	202.69	1	1	17	71.77	1	1	9.2	1	1
Grid	70	329.64	1	1	20	92.84	1	1	11.23	1	1
$\lvert\mathcal{G}\rvert = 7.5$	100	435.6	1	1	18	90.22	1	1	13.07	1	1
Intru	10	26.29	1	1.8	18	62.38	1	1.8	3.69	1	2.2
sion	30	73.08	1	1.13	19	142.63	1	1.13	4.09	1	1.13
Detec	50	103.58	1	1	20	194.55	1	1	4.44	1	1
tion	70	188.44	1	1	21	223.97	1	1	4.96	1	1
$\lvert\mathcal{G}\rvert = 15$	100	179.41	1	1	21	224.96	1	1	5.94	1	1
	10	120.94	0.9	2.3	21	215.32	0.9	2.3	4.35	0.6	1.8
Logis	30	1071.91	1	1.07	22	236.29	1	1.07	4.55	0.87	1.13
tics	50	813.36	1	1.2	23	238.87	1	1.2	5.37	1	1.2
$\lvert\mathcal{G}\rvert = 10$	70	606.87	1	1	24	243.38	1	1	6.29	1	1
	100	525.44	1	1	24	247.04	1	1	8.34	1	1
	10	0.67	0.93	1.33	10	0.97	0.93	1.33	0.74	0.67	1.27
	30	0.92	1	1	11	1.13	1	1	0.74	0.8	1.07
campus	50	1.11	1	1	11	1.31	1	1	0.77	0.8	1.13
$\lvert\mathcal{G}\rvert = 2$	70	1.41	1	1	11	1.63	1	1	0.8	0.8	1
	100	1.56	1	1	11	1.84	1	1	0.82	1	1.2
	10	77.85	0.88	1.25	11	80.74	0.88	1.25	1.55	0.88	1.25
	30	144.58	0.93	1.21	11	80.82	0.93	1.21	0.67	0.93	1.21
Kitchen	50	218.51	1	1.33	11	80.86	1	1.33	0.71	1	1.27
$\lvert\mathcal{G}\rvert = 3$	70	245.88	1	1.2	11	80.86	1	1.2	0.73	1	1.47
	100	488	1	1.47	12	81.16	1	1.4	0.82	1	1.6

Table 4.12: Evaluation with optimal and two satisficing planners. Each row describes averages over 15 different plan recognition problems. The columns stand for % of actions in plan for hidden goal sampled, average time in seconds for each complete plan recognition problem (T), average quality measuring fraction of problems where hidden goal is among the most likely (Q), average number of most likely goals (S).

The columns in Table 4.12 for HSP^*_f express the "normative" results, regarding how robust is the $P(G|O)$ distribution obtained, since it is derived from the optimal costs $c^*_{P'}(G+O)$ and $c^*_{P'}(G+\overline{O})$. Column T in all cases shows the *average time* per plan recognition problem. These times are larger for the optimal planner, approaching the 240 seconds time limit for the anytime planner, and are lowest for the greedy planner. For example, the time of 53 seconds reported in the third row for greedy LAMA, means that 53 seconds was the average time over the 15 Block Word *plan recognition problems* that resulted from sampling 50% of the observations. Since the problem involves 20 possible goals, we can deduce that the time to obtain $P(O|G)$ for each of the 20 possible goals was of about 2.6 seconds. The more accurate $P(O|G)$ estimate obtained from anytime LAMA is of 11 seconds, and getting the exact $P(O|G)$ requires 5 times as much time, about 60 secs.

On this same set of plan recognition problems anytime LAMA exhausts the allotted time, which points to the planning task being significantly challenging for LAMA planner heuristics and search strategy. If we look at the times reported by anytime LAMA on the IPC–GRID domain we can see that they are well below the 240 seconds deadline. This means that LAMA obtained an *optimal* plan, therefore obtaining exactly the same $P(G|O)$ as HSP$^*_f$ does. This observation is very relevant for engineers developing applications of plan recognition using our approach, since it makes evident the necessity to calibrate state–of–the–art satisficing classical planners on the action theory corresponding with their application domain. It is indeed possible that there is a planner which really works well, obtaining optimal plans with significantly less effort than optimal planners, performing the computation of solutions to the $P[G]$ planning problems which are used to derive $P(G|O)$.

The column L displays average optimal plan length, an statistic which helps to visualize how much information is conveyed by observation sequences. For example, observation sequences obtained from sampling up to 10% of the actions in optimal plans, might include as few as 2 actions. This information can well be insufficient to identify the true hidden goal with certainty, due to the ambiguous nature of the actions observed, as in the case of LOGISTICS plan recognition problems where it is just observed a truck moving in and out of a location where packages relevant to several goals lie.

Columns Q and S provide information about the quality of the solutions by focusing on the goals found to be the most likely, the set:

$$\mathcal{G}^S \stackrel{\text{def}}{=} \operatorname*{argmax}_{G \in \mathcal{G}_T} P(G|O)$$

Q is measuring how often the true hidden goal used to generate observations O was among the goals in $\mathcal{G}^S$. So we can see that the $P(G|O)$ elucidated by an optimal planner rarely – 25% of the time for the IPC–GRID problems with the shortest observation sequences – fails to include the true hidden goal in $\mathcal{G}^S$. S is reporting the size of the set of most likely goals, and is useful to determine how similar is the likelihood of plans for different goals given an observation sequence. When Observation sequences O are very short and plans for $\mathcal{G}_T$ share many actions – an event common in BLOCKWORDS since the initial configuration of the blocks in the table is the same for all goals – such sequences might be ambiguous enough to deem – incorrectly – as the most likely goal another than the true one.

We want to note that it is not always the case that the hidden goal used to generate the observations will turn out to be the most likely given the observations, even when O is compatible with an optimal plan for G. Indeed, if there are optimal plans for G that do not agree with O, and there is another achievable goal G' that has *not* such optimal plans, then in the formulation above, $P(G'|O)$ will be higher than $P(G|O)$. This is entirely reasonable, as G' is then a perfect predictor of O, while G is not.

4.12 Summary

This chapter contains the primary contribution of the thesis, namely, a formulation of a model–based, generative approach to plan recognition. Plan recognition problems are formalized as plan recognition *theories* T whose solution is a posterior probability distribution $P(G|O)$ over goals G in the hypothetical goal set $\mathcal{G}_T$ considered by theory

T. In order to obtain $P(G|O)$ we approximate the likelihood $P(O|G)$ of observations O for each of the possible hypothetical goals G, as a function over the costs of optimal plans for planning problems $P[G]$ that either satisfy or do not satisfy the observation sequence O. These costs are obtained by performing calls on optimal planners, without needing to modify the planner code in any way.

Along with the formulation and computational framework, several plan recognition tasks are discussed in depth. The flexibility granted by planning languages to model interesting and challenging plan recognition problems is made apparent, and is also shown how to convert plan libraries described in the literature into planning domains. Finally, experimental results are discussed that show that our approach scalability is directly tied with that of the planners used.

Chapter **5**

PR over POMDP Theories

In this chapter we will introduce the extension of the model–based framework for plan recognition presented in the previous chapter to a more expressive class of models, Goal Pomdps. We will start by showing a motivational example in a very simple task, where the observed agent has partial information on world state and effects of actions are not deterministic. Then we will discuss how Goal Recognition can be performed on the Goal Pomdp setting when the we have *full access* to all the actions performed by the observed agent when pursuing a given goal, thus extending the work in (Baker et al., 2009) which applies to the less expressive Goal Mdp models. After that, we will show how the posterior goal distribution $P(G|O)$ can be obtained from an observation sequence which contains only part of the actions executed by the observed agent. Then we will introduce the Pomdp tasks where we test our Goal Recognition scheme over partial observation sequences, based on sampling executions of a soft–max policy π defined over optimal value functions V^*. The robustness and efficiency of this method is tested empirically over several plan recognition tasks.

5.1 Motivation

As an illustration of how a plan recognition problem can be naturally cast in the Goal Pomdp setting, let us consider an agent that is looking for an item A or B each of which can be in one of three drawers 1, 2, or 3, with probabilities $P(A@i)$ and $P(B@i)$ equal to:

$$P(A@1) = 0.6\ , P(A@2) = 0.4\ , P(A@3) = 0$$
$$P(B@1) = 0.1\ , P(B@2) = 0.6\ , P(B@3) = 0.3$$

where $A@i$ denotes the fact "item A is in drawer i".

The actions available to the agent are to *open* and *close* the drawers, to *look for* an item inside an already open drawer, and to *grab* an item from a drawer if it is known to be there. When the agent actively looks for a certain item in a drawer, there is a probability of 0.2 that she fails spotting the item when it actually is inside that drawer. The probability of observing an item inside a drawer when that item is not inside the drawer is 0. No more than one drawer can be open at all times.

The hypothetical goal set $\mathcal{G}_T$ consists of three goals, G_1, G_2 and G_3, which correspond with having item A, item B and both. The prior probability distribution $P(G)$ over goals $G \in \mathcal{G}_T$ is

$$\begin{aligned} P(G_1) &= 0.4 \\ P(G_2) &= 0.4 \\ P(G_3) &= 0.2 \end{aligned}$$

meaning that it is considered to be equally likely that the agent is interested in either item, while the hypothesis of the agent being interested in finding *both items* is considered to be less likely.

We want to find out the goal posterior probabilities when the behavior of the agent is partially observed. In our setting, the observer gets to see some of the actions done by the agent, but not necessarily all of them. The observer must then fill up the gaps in order to deduce which goal G is the observed agent pursuing. Let us assume that it is observed that the agent opens drawer 1, then drawer 2, and then drawer 1 again. These correspond to the following observation sequence

$$O = (\ open(1),\ open(2),\ open(1)\ )$$

Under the assumption that the observed agent tries to minimize the number of actions executed while pursuing a goal, the most likely explanation for the observed actions O is that the agent is looking for item A. With this premise in mind, the choice the agent makes between opening either drawer is based on the *expected cost* to achieve its goal having doing so. Since the agent starts by looking in drawer 1, where the probability of finding B is 0.1, it makes little sense, if the number of actions is to be minimized, that she chose to open that drawer if B is the item being sought. This intuition is further reinforced by the fact that O starts and ends opening drawer 1. A likely explanation is that the agent did not observe A in that drawer, then closed it, and then looked for A in drawer 2. Then, probably the agent did not find A in drawer 2, and thus looked again in drawer 1, having to open it again.

Indeed, the algorithm that we will describe, concludes that the posterior probabilities for the three possible goals are $P(G_1|O) = 0.53$, $P(G_2|O) = 0.31$, and $P(G_3|O) = 0.16$, with G_1 as the most likely goal.

There are no other systems or formulations that can handle this type of scenarios where the agent gets partial observations from the environment, and the observer gets partial information about the actions of the agent, with some of the agent actions going possibly unnoticed.

5.2 Goal Recognition for Goal Pomdps

As in the fully–observable and deterministic setting described in the previous Chapter, the Goal POMDP describing the possible range of behaviors the observed agent can engage into, is assumed to be known by both the agent and the observer, except for the actual goal G of the agent. Instead, the set of hypothetical goals $\mathcal{G}_T$ is given along with the priors $P(G)$ for each goal G

Definition 5.1 (Plan recognition theory on a Goal POMDP setting)**.** *A plan recognition problem or* theory *is a triplet $T = \langle M[\cdot], \mathcal{G}, O, Prob\rangle$ composed by*

- *A* GPT *Goal* POMDP *domain $M[\cdot] = \langle F, A, I\rangle$,*
- *a* set of hypothetical goals *$\mathcal{G}_T$, each $G \in \mathcal{G}_T$ where G are formulas over functional symbols in F,*
- *an* observation sequence *O, where each $o_i \in O$ corresponds with some action $a \in A$,*
- *and a* prior probability *distribution $Prob$ defined over $G \in \mathcal{G}_T$.*

which is similar to Definition 4.1 with the provision of substituting the STRIPS domain $P[\cdot]$ by a GPT Goal POMDP planning domain $M[\cdot]$[1].

As for plan recognition theories in the classical setting, the solution to a plan recognition theory T in the Goal POMDP setting is a posterior goal probability distributions $P(G|O)$, where goals $G \in \mathcal{G}_T$ are ranked according to how well do they predict observation sequence O. Posterior goal probabilities $P(G|O)$ are obtained from the Bayes rule:

$$P(G|O) = \alpha P(O|G)P(G) \tag{5.1}$$

where α is a normalizing constant that does not depend on G. The problem of inferring the posteriors $P(G|O)$ gets thus mapped into the problem of defining and computing the likelihoods $P(O|G)$.

The key assumption is that if the agent is pursing goal G, the probability $P(a|b, G)$ that she will choose action a in the belief state b is given by the Boltzmann policy:

$$P(a|b, G) = \alpha' exp\{\beta\, Q_G(a, b)\} \tag{5.2}$$

where α' is a normalizing constant and β captures the principle of rationality (Dennett, 1983; Baker et al., 2009) in a *soft* way. For large values of the β parameter, the agent chooses her actions minimizing the expected cost Q_G to achieve goal G, and indeed acts optimally when Q_G is optimal. For low β values, the agent selects actions in a random basis, paying little heed to the information about the merit of doing a encoded by Q_G.

The term $Q_G(a, b)$ expresses the expected cost – see Definition 3.3 – to reach the goal G from b, when the action a executed while being in the belief state b, is given by the following formula

$$Q_G(a, b) = c(a, b) + \sum_{\omega \in Obs} b_a(\omega) V_G(b_a^\omega) \tag{5.3}$$

V_G stands for the solution to the Bellman equation for the belief b – see Equation 3.8 – assuming that the goal states are those in which G is known to be true with certainty and The term $c(a, b)$ denotes the expected cost of action a in b. $b_a(\omega)$ and b_a^ω, which were defined in Equation 3.5, stand for the probability that agent obtains observation token ω after doing action a in b, and the probability distribution that results from doing a in b and actually getting observation token ω.

We will leave the definition of observation sequences O to Sections 5.3 and 5.4 where we offer two different accounts of goal recognition depending on the assumptions

[1]See Definition 3.19.

made regarding missing actions and the availability or not of observation tokens ω to the observer.

5.3 Complete Observation Sequences

When discussing plan recognition in the classical setting on Chapter 4 we only explicitly addressed the problem of handling *incomplete* observation sequences O. This is indeed a more general problem than that of considering just the *prefix* of some classical plan, and the results presented in Chapter 4 do indeed apply, without further work, for that case.

We will however specifically address complete observation sequences when moving into the Goal POMDP setting, since it provides us with the opportunity of generalizing a recent (Baker et al., 2009) account of goal recognition for the Goal MDP setting which makes two assumptions which limit its flexibility. One is that of requiring observation sequences O to be complete, and the other, that of assuming that the states of the MDP model resulting from the observed agent performance, are *fully observable* for both the observer and the agent.

Both assumptions result in a pretty restrictive framework, but serve to reduce the goal recognition problem to a simple probabilistic inference problem. In this section, we take (Baker et al., 2009) into a Goal POMDP setting, thus gaining a great deal of flexibility. The observed agent now is modeled as having *partial* knowledge of the world state s, modeled with a *belief state* – see Section 3.4 – a probability distribution b over possible states s.

The knowledge the agent has about the actual state s is modified by those observation tokens she gathers while doing actions. We assume that these tokens ω are available to the observer along with executed actions, as per Definition 5.2. This is a critical yet subtle difference, since rather than requiring the full description of state s, only the feedback *actually* gathered by the observed agent after doing an action is needed to characterize the likelihood $P(O|G)$ of observing sequence O while pursuing goal G.

Hence observation sequences are defined in order to account for the partial knowledge the observed agent has of its environment and how the actions done change it:

Definition 5.2 (Complete observation sequences). *A* complete *observation sequence O in the Goal* POMDP *setting is a sequence*

$$O = (\langle a_1, \omega_1 \rangle, \ldots, \langle a_i, \omega_i \rangle, \ldots, \langle a_n, \omega_n \rangle)$$

of n pairs $\langle a_i, \omega_i \rangle$, where a_i is an action in the planning domain $M[\cdot]$ action set A, and ω_i is the observation token belonging to the Obs set prescribed by $M[\cdot]$, obtained after doing action a_i.

Implicit in this definition is the assumption that the observer receives exactly the same *feedback*, in the form of observation tokens ω_i, the observed agent got from doing actions a_i.

In order to obtain the likelihood $P(O|b, G)$, for the *complete* observation sequence $O = (\langle a_1, \omega_1 \rangle \ldots, \langle a_i, \omega_i \rangle \ldots, \langle a_n, \omega_n \rangle)$, we just need to rely on Equations 3.5 which describe how the agent initial belief state b_0 changes as actions a_i are executed and observation tokens ω_i are retrieved, resulting in the sequence of beliefs $b_{a_i}(\omega_i)$. Since consecutive beliefs $b_{a_i}^{\omega_i}$) are independent, $P(O|b, G)$ follows from this recursive decomposition

$$P(O_{i,n}|b_i, G) = P(a_i|b_i, G)\, b_{a_i}(\omega_i)\, P(O_{i+1,n}|b_{a_i}^{\omega_i}, G) \tag{5.4}$$

where $O_{i,n}$ is the subsequence of O which comprises actions $a_i, \ldots, a_n$, a_i is the i-th action executed, b_i is the belief state resulting from executing action a_{i-1}, $b_{a_i}(\omega_i)$ being the probability of ω_i being the token observed for belief b_i and executing a_i. $b_{a_i}^{\omega_i}$ is the belief that results from integrating the knowledge about action a_i effects and the information conveyed by observation token ω_i.

We then obtain the desired posterior goal probabilities $P(G|O)$ by setting the likelihood $P(O|G)$ to $P(O_{1,n}|b_0, G)$, and plugging it into Equation 5.1.

The POMDP planner enters into this formulation by providing the expected costs $V_G(b)$ to reach G from b, that are used via the factors $Q_G(a, b)$ for defining the probability that the agent will do action a when in belief state b. This probability distribution is given by the *policy* the observed agent is assumed to be following. One possible example is the Boltzmann policy depicted in Equation 5.2, but it could well be any other policy the plan recognition system designer might find suitable for the application at hand.

5.4 Incomplete Observation Sequences

In the account given in Section 5.3, the information conveyed by the observation sequence O that is available to the observer describes both the actions done by the agent, and the observation tokens that the agent receives from the environment. Moreover, this sequence of actions and tokens is assumed to be complete, meaning that no one is missed by the observer. In this section we will discuss a formulation of goal recognition where these two restrictions are lifted.

In this formulation we use the definition we gave of observation sequences O for the classical setting – see Section 4.1 – that is, the observation sequence O is a sequence of actions $a_1, \ldots, a_n$ which is *not necessarily* complete. It is assumed that some of the agent actions may have gone unnoticed to the observer, who cannot then assume a priori that action a_{i+1} is the action that the agent did right after a_i. Besides that, the observation tokens gathered by the observed agent are no longer available.

Still, as before, the posterior goal probabilities $P(G|O)$ can be derived using Bayes rule (5.1) from the priors $P(G)$ and the likelihoods $P(O|G)$ that can now be defined as

$$P(O|G) = \sum_{\tau} P(O|\tau)P(\tau|G) \tag{5.5}$$

where τ ranges over the possible policy executions that arise when the agent pursues goal G and chooses actions a according to some policy π with probability $P(a|b, G)$. Executions $\tau = (a_1, \ldots, a_i, \ldots, a_n)$ contain the complete sequence of agent actions, from the initial belief b_0 until a belief state b where the goal is a certain known property of the underlying hidden state s.

In order to obtain $P(O|\tau)$ we rely on the notion of action sequences satisfying observation sequences O presented in Definition 4.2 so that $P(O|\tau)$ is defined as:

$$P(O|\tau) = \begin{cases} 1 & \text{if } \tau \text{ satisfies } O \\ 0 & \text{otherwise} \end{cases} \tag{5.6}$$

which follows from the fact that executions τ which do not have O embedded cannot be generating it.

5.5 Computation of Observation Likelihoods

With the definition of $P(O|\tau)$ above, then the sum in Equation 5.5 can be approximated by *sampling* executions τ from distribution $P(\tau|G)$ and counting the frequency the executions obtained comply with the observation sequence O

$$P(O|G) \approx m_O/m \tag{5.7}$$

where m is the total number of executions sampled for each goal $G \in \mathcal{G}_T$, and m_O is the number of such executions that complies with O.

In order to sample $P(\tau|G)$ it is required to identify which is the policy π the agent is actually using. This is an application specific problem, and we conduct our experiments assuming the agent follows the Boltzmann policy described in Equation 5.2. The policies particular to each goal $G \in \mathcal{G}_T$ are furnished by the POMDP planner, that computes the value functions $V_G(b)$ from the Goal Pomdp problem $M'_{Obs}[G]$ that results from composing each goal with the domain $M[\cdot]$ specified.

Executions are then sampled by performing a policy *roll-out* or simulation, a two–step procedure that involves first sampling action a with probability $P(a|b, G)$, and second, to sample the observation token ω that results from executing the sampled action a on the current belief state b according to the distribution $b_a(\omega)$. This second step is required since in this formulation the observation tokens gathered by the agent are no longer assumed to be available in O.

The process described above is applied on the initial belief b_0, provided by the domain description $M[\cdot]$, iteratively until a belief state b is found such that it can be concluded that a goal state $s' \in S_G$ has been reached with certainty e.g. the probability $b(s')$ assigned by belief b to state s' is 1.

The result of the roll-out procedure is the trajectory followed across belief space

$$b_0, a_0, \omega_0, b_1, a_1, \omega_1, \ldots$$

where $b_{i+1} = b_a^\omega$ for $b = b_i$, $a = a_i$ and $\omega = \omega_i$, actions a_i and observations tokens ω_i have been sampled with probability $P(a_i|b_i, G)$ and $b_a(\omega_i)$ respectively, for $b = b_i$ and $a = a_i$. Executions τ are the actions a_i appearing on this trace.

Once m executions τ have been obtained, then the likelihood $P(O|G)$ is computed through Equation 5.7 and plugged into Equation 5.1 from which the posterior goal probabilities $P(G|O)$ are obtained.

In order to analyze later in Section 5.7 this formulation, we characterize the goal recognition system as a *binary classifier* $\mathrm{GR}^{m,\beta}$ that maps goals $G \in \mathcal{G}_T$ into labels

T, F where T stands for “true hidden goal” and F for “false hidden goal”. This classifier requires parameters m and β, which correspond to the number m of executions τ being sampled for each goal G and β parameter of the Boltzmann policy. The influence of both parameters in the accuracy and robustness of this formulation are evaluated in Section 5.7 over three benchmarks which are described in Section 5.6.

Goal recognition system designers, besides considering what policy and selection of parameter m suits their application best, should take into account as well that a substantial part of the computational cost of this account can be performed in an off–line manner. Value functions V_G over beliefs can be precomputed and stored so they are readily used in *on–line* estimation of likelihoods $P(O|G)$ following Equation 5.7 by gathering simulated executions sampled from the probability distribution that results from the policy they choose to use.

5.6 Evaluation Domains

We have designed three different Goal POMDP domains in order to test the approach to goal recognition over incomplete observation sequences presented on Section 5.4. They cover three possible plan recognition tasks similar to those presented in Chapter 4. All three domains involve the need of reasoning about feedback obtained by doing information gathering actions, as well non–deterministic effects. OFFICE and KITCHEN cover situated agents that need to both navigate across a map and interact with objects in diverse complex ways in order to achieve some goal. DRAWERS, on the other hand, is a simple task that features in a clear way the possibilities of our proposed formulation for goal recognition.

Name	$\lvert S\rvert$	$\lvert A\rvert$	$\lvert Obs\rvert$	$\lvert b_0\rvert$	$\lvert\mathcal{G}\rvert$	$\lvert T\rvert$
OFFICE	2,304	23	15	4	3	3.4
DRAWERS	3,072	16	16	6	3	4.5
KITCHEN	69,120	29	32	16	5	10.1

Table 5.1: Features of the four domains: number of states, actions, observation tokens, states in initial belief, and possible goals. T is time in seconds to compute $V_G(b_0)$ for all goals G in $\mathcal{G}$.

Table 5.1 shows the number of states, actions, observations, goals, and possible initial states for each of the three domains we have used to evaluate the proposed goal recognition scheme described on Section 5.4. These problems are relatively easy to solve for state–of–the–art Goal POMDP solvers, but possess some of the features that make POMDPs an interesting model for goal recognition. Namely, uncertainty in the initial state, actions with noisy effects and noisy observations. We will describe the domains next.

Drawers

DRAWERS is the task we discussed in the Motivation Section at the start of the chapter.

Signature	Domain	Observable
loc(i)	$\{hand, d1, d2, d3\}$	No
open(d)	$\{T, F\}$	Yes
carries(i)	$\{T, F\}$	Yes
sees(i)	$\{T, F\}$	Yes

Table 5.2: DRAWERS state variables. *hand* is the object denoting the observed agent hand, $d1$, $d2$ and $d3$ denote drawers. T and F denote the boolean constants True and False.

World state is represented with the fluents shown on Table 5.2. The fluents $loc(i)$ that denote the actual location of items A or B are hidden from the agent, since these are randomly tossed at the start into drawers according to the probability distribution detailed in Section 5.1. Fluents $carries(i)$ and $sees(i)$ are used to represent the knowledge the agent actually has about item locations. Note that only actions whose preconditions are *known* to be true, i.e. $b(s) = 1$ for s where the precondition holds, can be executed.

The agent hypothetic goals $\mathcal{G}_T$ are

$$\begin{aligned} G_1 &: \quad \text{carries}(A) \\ G_2 &: \quad \text{carries}(B) \\ G_3 &: \quad \text{carries}(A) \wedge \text{carries}(B) \end{aligned}$$

that is, to be holding either an object named A, an object named B or both.

We consider both singleton goals to be random independent events, with equal prior probability, and we set the probability of the joint goal to $P(holding\ A)P(holding\ B)$. This follows from $P(A \cap B) = P(A)P(B)$ when A and B are considered to be random independent events.

Signature	Precondition	Effects	Obs
open(d)	open(d) $= F$	open(d) $= T$	None
close(d)	open(d) $= T$	open(d) $= F$	None
pick(d, i)	open(d) $= T$ sees(i) $= T$	loc(i) $= d \rightarrow$ carries(i) $= T$	carries(i)
lookUp(d, i)	open(d) $= T$	loc(i) $= d \rightarrow$ sees(i) $= T$ [0.7]	sees(i)

Table 5.3: DRAWERS observed agent actions. Preconditions with more than one fluent are assumed to be conjunctions unless noted otherwise. Effects of the form $x \rightarrow y$ denote a conditional effect. When $[p]$ appears next to an effect means that the specified effect is an outcome of the action with probability p. All actions have the same cost, which is set to 1.

The actions available to the observed agent are described on Table 5.3. The most interesting action is lookUp(d, i), that the observed agent can use to search for item i inside drawer d. The action effect consists of one single *conditional* effect, which makes visible to the agent with probability 0.7 item i, when that item is indeed inside drawer d. If this is the case, action lookUp emits the observation token sees(i) $=$ T. This effect is implicitly defining the sensor model probabilities $Q(o|s, a)$. The probability of actually getting the observation $b_a(\omega)$ is given by the probability $b(s)$ the current belief b assigns to a state s where item i is inside drawer d and $Q(o|s, a)$.

Office

Office is adapted from the plan recognition task discussed in (Bui, 2003). In our version the observed agent is at a lab which consists of two rooms: one is her office, where she has a workstation and a cabinet to store her coffee cup and blank spare paper for printing. The other is the club, where the coffee machine and printer are placed. The two rooms are connected by a corridor. The agent goals might be to print an article, have a cup of coffee or accomplish both things.

The agent is initially at the corridor and knows her own location. However, the agent does not know about the initial status of the printer which can be either out of paper, clogged, both or none. To print an article, the agent needs to get to the workstation in her office, send the file to the printer queue and then get to the printer to retrieve it. The outcome of this sequence of actions depends on the initial state of the printer and also on whether the printer clogs itself when printing the paper. The unreliability of the printer is modeled by assigning two possible outcomes to print commands, with the outcome of the printer getting clogged having probability 0.2. If the printer was initially clogged or out of paper, the observed agent will need to either service it by removing the paper clogging the printer paper feed, or to go back to her office to fetch blank paper.

Table 5.4 shows the functional fluents used to model states. Two types are used in our modeling, *item* and *object*. *Items* are those objects the observed agent can carry, an event which is represented with carries(i) functional fluent being true. *Items* in our benchmark include a *mug*, a slab of *blank paper*, and an *article* printout. *Objects* is used to denote those objects the observed agent cannot carry but can interact with provided she is in front of them, represented with the inFrontOf(o) fluent. These consist of the *printer*, the agent's *computer* and office *cabient*, and the $coffee$ machine.

Signature	Domain	Observability
carries(i)	$\{T, F\}$	Yes
at()	$\{office, corr, club\}$	Yes
inFront(o)	$\{T, F\}$	Yes
outOfPaper()	$\{T, F\}$	Initially Unknown
clogged()	$\{T, F\}$	Initially Unknown
available()	$\{T, F\}$	Yes
queued()	$\{T, F\}$	Yes
hasCoffee()	$\{T, F\}$	Yes
drank()	$\{T, F\}$	Yes
read()	$\{T, F\}$	Yes
queued()	$\{T, F\}$	Yes

Table 5.4: Office state variables. Fluent meaning is described in the text. i denote arguments of type *item* and o arguments of type *object*. *corr* stands for the corridor.

The agent location is described by its general location – the *at*() fluent value – and whether it is in a position inFront(o) which allows her to interact with objects o in that location. Finally, outOfPaper() and clogged() are used to track the state of the printer, while available() and queued() represent whether the paper has been already printed or is waiting to be processed in the printer's queue.

We note that the initial belief b_0 for the resulting Goal POMDP is a probability distribution $b(s)$ over four possible states s all of them with probability 0.25. Each possible initial state corresponds with one of each possible combinations of clogged() and outOfPaper() values, denoting the situations where the printer is neither clogged or out of paper, it is in either condition or suffers from both conditions at the same time. at() = *corr* initially and all other fluents are false.

The hypothetical goals $\mathcal{G}_T$ are then defined as:

$$\begin{aligned} G_1 : &\quad \text{read}() = T \\ G_2 : &\quad \text{drank}() = T \\ G_3 : &\quad \text{read}() = T \wedge \text{drank}() = T \end{aligned}$$

that is, that the observed agent wants to read the paper, wants to drink some coffee or desires to do both.

Signature	Precondition	Effects	Obs
walk($lab, corr$)	at() = lab	at() = $corr$ inFront($computer$) = F inFront($cabinet$) = F	None
walk($corr, lab$)	at() = $corr$	at() = lab	None
walk($corr, club$)	at() = $corr$	at() = $club$	None
walk($club, corr$)	at() = $club$	at() = $corr$ inFront($printer$) = F inFront($coffee$) = F	None
walkTo($computer$)	at() = lab	inFront($cabinet$) = F inFront($computer$) = T	None
walkTo($cabinet$)	at() = lab	inFront($computer$) = F inFront($cabinet$) = T	None
walkTo($coffee$)	at() = $club$	inFront($printer$) = F inFront($coffee$) = T	None
walkTo($printer$)	at() = $club$	inFront($coffee$) = F inFront($printer$) = T	None
takeMug()	at() = lab inFront($cabinet$) = T	carries(mug) = T	None
takePaper()	at() = lab inFront($cabinet$) = T	carries($paper$) = T	None
checkPaper()	at() = $club$ inFront($printer$) = T		outOfPaper()
openCanopy()	at() = $club$ inFront($printer$) = T		clogged()
getArticle()	at() = $club$ inFront($printer$) = $true$ available() = T	carries($article$) = T	None
readArticle()	carries($article$) = T	read() = T	None
pourCoffee()	at() = $club$ inFront($coffee$) = T carries(mug) = T	hasCoffee() = T	None
drinkCoffee()	hasCoffee() = T	drank() = T	None

Table 5.5: OFFICE observed agent simple actions. More complex actions are detailed in the text.

Table 5.5 details actions with simple deterministic effects, more complex actions are described next. The observed agent can submit the article to the printer with the action submitPrintJob(). The precondition for doing so requires her to be at the club and in front of the computer, noted in GPT notation below:

$$\text{at}() = \textit{lab} \wedge \text{inFront}(\textit{computer}) = T$$

When executed, the action has two possible outcomes. With probability 0.2 the printer gets clogged while trying the print the article, though the print job remains in the queue. This outcome is encoded by setting to true the relevant fluents:

$$\text{queued}() = T$$
$$\text{clogged}() = T$$

With probability 0.8, when the printer is not clogged nor out of paper, the article becomes available to be picked up at the printer tray. This is encoded with the following conditional effect:

$$\text{outOfPaper}() = F \wedge \text{clogged}() = F \rightarrow \text{available}() = T$$

In the case that the above is not true in the hidden state s, the article is not printed but remains in the queue:

$$\text{outOfPaper}() = T \vee \text{clogged}() = T \rightarrow \text{queued}() = T$$

Note that the agent is always aware whether the article got printed or not, which gives her a hint that further actions are needed to get it printed.

These actions involve servicing the printer by either reloading its blank paper tray or to unclog the blank paper feed. The first activity corresponds with the action reloadPaperTray(), with precondition:

$$\text{outOfPaper}() = T \wedge \text{inFront}(\textit{printer}) = T \wedge \text{carries}(\textit{paper}) = T$$

In order to execute this action, the agent needs first to check the tray itself, by executing the information gathering action checkPaper() that reveals the value of outOfPaper() fluent. If the conditions above are met, then the article gets printed provided it was already waiting on the queue and the printer is not clogged:

$$\text{clogged}() = F \wedge \text{queued}() = T \rightarrow \text{available}() = T$$

Unclogging the printer is achieved with the action servicePrinter(), that requires the agent to have previously checked the paper feed with the information gathering action openCanopy()

$$\text{clogged}() = T \wedge \text{inFront}(\textit{printer}) = T$$

which establishes the truth value of fluent clogged(). When executed, servicePrinter() will make the article available, provided the printer is not out of paper and the article was in the queue:

$$\text{outOfPaper}() = F \wedge \text{queued}() = T \rightarrow \text{available}() = T$$

GPT modeling language allows us to express in a compact form a simple yet lively task by combining probabilistic and conditional effects. We note that the best, as in minimizing expected cost, policy for goal G_1 starting from the initial belief consists in going directly to the computer, submitting the job and then fetching paper from the cabinet *in case* the printer is out of paper. The executions resulting from executing this policy can overlap in many ways with the best policy for G_2, which only uses deterministic actions.

Kitchen

The KITCHEN plan recognition task is inspired in the domain discussed by (Fern et al., 2007) in the context of using Goal POMDPS as a computational framework to develop *helper agents* that assist humans in a interactive way to make easier everyday tasks.

Here the observed agent is a person who is known to be interested in cooking one of four possible dishes. Each dish requires up to 3 ingredients out of a possible 4 given ingredients i_1, i_2, i_3, i_4, and mix them in a bowl. Ingredients are known to be inside cupboards, though the precise locations of the ingredients is not known to the agent – nor the observer.

Besides getting a hold of the ingredients, the observed agent some of the dishes require to cook the ingredients in the bowl by either boiling or frying the mix and possibly having to both as well. One of the dishes does not require the ingredients to be cooked, but have to be served on a tray. For this she needs to get the required kitchenware from a third cupboard, and put them on the stove. In order to obtain

Signature	Domain	Observability
container(i)	$\{0, 1\}$	Observable
inside(i)	$\{T, F\}$	Initially Unknown
location()	$\{0, \ldots, 4\}$	Always
onHand()	$\{0, \ldots, 7\}$	“
onStove()	$\{0, \ldots, 3\}$	“
capacity()	$\{0, \ldots, 3\}$	“

Table 5.6: KITCHEN state variables. Fluent meaning is described in the text. i denote ingredients being taken as arguments by the fluent.

a compact Goal POMDP for this task, we have used the fluents shown on Table 5.6. The cupboards, the bowl and the stove ovens are modeled as discrete locations. We have modeled these locations as integers, mapping each of the three cupboard into the range $[0, 2]$, while 3 and 4 represent the location of the bowl and that of the stove, respectively.

Ingredients initial location is modeled with the inside(i) fluents, where the integers 0 and 1 identify the cupboards containing them. Kitchenware location is not modeled explicitly by one single fluent. Rather than that, we have chosen to model it with the onStove() and onHand() fluents, whose values correspond to either kitchenware on the stove – 1 for the pan, 2 for the pot, 3 for both, 0 to denote that no piece of kitchenware lies on the stove – or which are being carried on her hands by the agent – 5 for the pan, 6 for the pot and 7 for the tray. If no kitchenware is on the stove or being carried by the agent, then it is in the third cupboard, since the agent is not able to carry more than one item on his hands.

The fluent capacity() models the limited volume of the bowl, so that at most 3 different ingredients can be in it. This allows us to constraint significantly the possible observed behaviors and to showcase a practical example of the succint manner in which GPT supports actions affecting numerical quantities.

The initial belief b_0 comprises 16 different possible states s, corresponding to each of the permutations of ingredients and cupboards. All possible states s have the same probability $b_0(s)$. The hypothetical goal set $\mathcal{G}_T$ consists of 4 goals, one for each possible dish. Goals are encoded as shown below, replacing integers by the names of the objects for the sake of clarity:

$$\begin{aligned}
G_1: &\quad \text{inside}(i_4) = T \wedge \text{onHand}() = \textit{tray} \wedge \text{position}() = \textit{bowl}\\
G_2: &\quad \text{inside}(i_1) = T \wedge \text{inside}(i_3) = T \wedge \text{onStove}() = \textit{pan\&pot}\\
G_3: &\quad \text{inside}(i_2) = T \wedge \text{onStove}() = \textit{pot}\\
G_4: &\quad \text{inside}(i_1) = T \wedge \text{inside}(i_3) = T \wedge \text{onStove}() = \textit{pan}
\end{aligned}$$

These terminal situations correspond with the precondition of a goal–specific action that sets a dummy fluent to signal that the goal has been achieved. We note that the confection of each dish requires executing actions so that observation sequences conveying incomplete executions can lead to situations where $P(G|O)$ can be very similar for two different goals G_i and G_j. For instance, achieving G_3 shares with G_2 that the agent will have to eventually reach for the pot and put it on the stove.

Signature	Precondition	Effects	Obs
moveTo(l)	position() $\neq l$	position() $= l$	None
inspect(c)	position() $= c$		container(i_1) $= c$ container(i_2) $= c$ container(i_3) $= c$ container(i_4) $= c$
pick(i, c)	position() $= c$ container(i) $= c$ onHand() $=$ *empty*	onHand() $= i$	None
take(k)	position() $= c_3$ onHand() $=$ *empty*	onHand() $= k$	None
putAway(k)	position() $= c_3$ onHand() $= k$	onHand() $=$ *empty*	None
putStove(k)	position() $=$ *stove* onHand() $= k$	onHand() $=$ *empty*	None
removeStove(k)	position() $=$ *stove* onHand() $=$ *empty*	onHand() $= k$	None
pour(i)	position() $=$ *bowl* capacity() > 0 inBowl(i) $= F$	inBowl(i) $= T$ onHand() $=$ *empty* capacity() $=$ capacity() $- 1$	None
clearBowl()	capacity() < 3	capacity() $= 3$ inBowl(i_1) $= F$ inBowl(i_2) $= F$ inBowl(i_3) $= F$ inBowl(i_4) $= F$	None

Table 5.7: Actions available to the observed agent in the kitchen KITCHEN task.

Actions available to the observed agent are listed on Table 5.7. Inspecting the cupboards holding the ingredients is modeled with the inspect(c) action, where c denotes any of the two possible cupboards. This is an information gathering action, which produces the observation tokens container(i) $= c$ that allow the observed agent to learn whether ingredient i can be found inside cupboard c.

The agent can move freely between any of the locations modeled – cupboards, bowl and stove – doing the *moveTo*(l) actions, where l stands for some location. In order to avoid loops that severely impact Gpt performance when computing value function $V_G(b)$, we set a its precondition that the destination is a different location than the one the agent currently is.

Ingredients i can be picked up from cupboards c with the action pick(i, c). In order to be possible to pick an ingredient, the agent needs first to have done inspect(c) on that cupboard as well as having her hands empty. Actions take(k), putAway(k), putStove(k) and removeStove(k) allow the agent to take and store kitchenware k into its cupboard and to put it, or remove it, on top of the stove in order to cook the ingredients required by a dish.

Once the agent gets hold of an ingredient i, she can pour them into the bowl with the action pour(i). In order to do so, the bowl needs to have spare room for the ingredient, and cannot contain it already, in order to avoid loops. The bowl has a limited capacity – in this particular task, it can hold up to three different ingredients – which is decreased each time the agent pours an ingredient into it. If for any reason, the wrong ingredient is poured into the bowl, the agent can restart the procedure, by doing the clearBowl() action, which removes all ingredients from the bowl and resets its capacity.

5.7 Experimental Results

For testing the goal recognition scheme proposed in Section 5.4, we used the Pomdp planner Gpt (Bonet and Geffner, 2001b) built around the Labeled Rtdp-Bel algorithm (Bonet and Geffner, 2003) and described on Section 3.3. The software for the goal recognition part was implemented on Python, making calls to Gpt when necessary. All the domains described in the previous Section, code and datasets discussed in this Section can be obtained from `https://sites.google.com/site/prasplanning`.

Experimental Dataset

The dataset we use in our experiments is made up of a set of instances, which are generated automatically and consist of pairs $\langle O, G \rangle$, where O is an observation sequence and G one of the hypothetical goals in $\mathcal{G}_T$. These pairs are obtained by applying the following process to each of the domains discussed in Section 5.6:

1. For each of the goals G in $\mathcal{G}_T$, the value function $V_G(b)$ is computed for the Goal Pomdp problem $P[G]$ with the Gpt planner.
2. 100 executions τ of the greedy policy over $V_G(b)$ – see Definition – are generated.
3. 10 executions are selected at random from the previous set, and 10 observation sequences O are obtained by selecting up to 30% of the actions in the sampled execution.
4. The procedure above is repeated, but changing the rate of actions selected to 50% and repeated again taking 70% of the observation sequences.
5. Each generated sequence O is paired with its goal G, yielding the pairs $\langle O, G \rangle$.

The true goal G is kept along O for validation purposes, and subsequently remains hidden to the goal recognition system.

Goal Recognition as a Binary Classification Problem

As discussed on Section 5.4 we pose the problem of goal recognition as that of labeling hypothetical goals $G \in \mathcal{G}_T$ in one of two classes, that of "true" hidden goals and that of "false" hidden goals, on the basis of the evidence supporting the hypothesis of O being the result of the observed agent pursuing hypothetical goals $\mathcal{G}_T$.

The classifier $\mathrm{GR}^{m,\beta}$ is then characterized by the equation below

$$\mathrm{GR}^{m,\beta}(G,O) = \begin{cases} T & \text{if } G = \mathrm{argmax}_{G' \in \mathcal{G}_T} P(G'|O) \\ F & \text{otherwise} \end{cases} \tag{5.8}$$

where m is the number of executions obtained by simulating the Boltzmann policy over $V_G(b)$ with parameter β.

Since $P(G|O)$ is a real valued function we set a tolerance threshold for the argmax function over floating point numbers, x and x', such that equality $x = x'$ is defined as follows

$$EQ(x,x',\epsilon) = \begin{cases} T & \text{if } |x - x'| < \epsilon \\ F & \text{otherwise} \end{cases} \tag{5.9}$$

In the experiments discussed next we set ϵ to 10^{-7} which, given the context of the computation, we consider to be adequate.

Performance Metrics

Rather than reporting posterior probabilities $P(G|O)$ for each hypothetical goal G, we report the performance of the classifier $\mathrm{GR}^{m,\beta}$ measured in terms of well–known classifier performance metrics used in the Machine Learning community (Fawcett, 2006) over a range of values for parameters m and β.

The basis of the performance metrics used rely on the notions of positive (P) and negative (N) classification tests. A goal recognition each instance $\langle O, G\rangle$ will be tested against each hypothetic goal G' and it will be deemed as *positive* whenever $\mathrm{GR}^{m,\beta}(G',O) = T$, i.e. G' is found to be the most likely goal that generated O, and negative when $\mathrm{GR}^{m,\beta}(G',O) = F$.

A positive test $\langle O, G\rangle$ for goal G' is a *true positive* (TP) if it is a positive instance and G' is equal to the actual hidden goal G, and is a *false positive* (FP) if not. Similarly, a negative test $\langle O, G\rangle$ for goal G' is *true negative* (TN) if G' is indeed different from the actual hidden goal G, and a *false negative* (FN) otherwise.

The classifiers $\mathrm{GR}^{m,\beta}$ are evaluated according to the standard metrics detailed below obtained from $\mathrm{GR}^{m,\beta}$ confusion matrix:

$$\begin{array}{rcl} TPR = & \frac{TP}{P} & \text{(True Positive Ratio)} \\ FPR = & \frac{FP}{N} & \text{(False Positive Ratio)} \\ ACC = & \frac{TP+TN}{P+N} & \text{(Accuracy)} \\ PPV = & \frac{TP}{TP+FP} & \text{(Precision)} \end{array}$$

The True Positive Ratio and False Positive Ratio allow to measure to what extent the classifier $\mathrm{GR}^{m,\beta}$ is reliable when it deems a hypothetical goal G' as the goal being pursued by the agent that generated observation sequence O. The relationship between these two measures allows to detect if the classifier is significantly better than deeming goals G' as true on a random basis, that entails that both TPR and FPR would have a value of 0.5. Accuracy assesses the overall reliability of $\mathrm{GR}^{m,\beta}$ for discriminating goals in $\mathcal{G}_T$ since it leverages true positive and true negative results simultaneously. Finally, precision is useful to detect when observation sequences O are ambiguous so many hypothetic goals are deemed as the true hidden goal.

Results

Figure 5.1 shows the aggregate results of the goal recognizer $\mathrm{GR}^{m,\beta}$ over the three benchmarks discussed in Section 5.6 in the form of a receiver operating characteristics (ROC) graph (Fawcett, 2006) that plots TPR against FPR. As it can be seen, the performance of the goal classifier approaches the optimal vertex $(1, 0)$ as the number of samples m becomes large. Performance is good – high TPR, low FPR – for $m \geq 1,000$ and high β values.

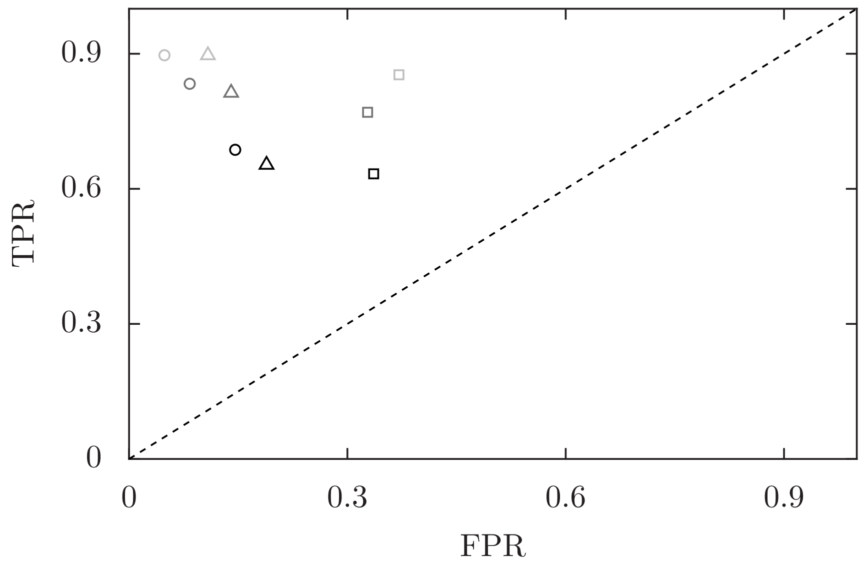

Figure 5.1: ROC graph showing aggregate results for goal classifier $\mathrm{GR}(m, \beta)$ for different m and β values (number of samples and noise level in action selection). Squares, triangles and circles denote different m values: 100, 1000 and 10000. Black, gray, and light gray denote different β values: 1, 10, 40. Results for random strategy represented by the dotted line.

Table 5.8 offers a detailed picture of the performance of the classifier $\mathrm{GR}^{m,\beta}$ setting m to $10,000$ and β to 40. We see across all domains how the accuracy of goal recognition increases as more information is conveyed by the input observation sequence, with near optimal results in all cases as the ratio of observation approaches 70%.

Processing one observation sequence involves simulating the Boltzmann policy m times for each goal, hence the similarity between the run-times for Office and Drawers, whose hypothetical goal sets $\mathcal{G}_T$ have the same size. However, the latter is somewhat more expensive since the agent might need to inspect a drawer several times until she finds the object, leading to longer executions.

Domain	Obs %	L	T	ACC	PPV	TPR
office	30	4.9	24.6	0.99	0.97	1.00
	50	7.6	24.7	1.00	1.00	1.00
	70	10.8	24.8	1.00	1.00	1.00
kitchen	30	3.8	95.2	0.86	0.73	0.73
	50	5.8	95.1	0.93	0.85	0.85
	70	8.3	95.2	0.98	0.95	0.95
drawers	30	2.9	38.8	0.84	0.77	0.77
	50	3.9	38.8	0.87	0.80	0.80
	70	6.0	38.8	0.96	0.93	0.93

Table 5.8: Performance of classifier GR($m = 10,000, \beta = 40$): domains, observation ratio, average length of observation sequences (L), average time in seconds to process O (T), average accuracy (ACC), precision (PPV) and True Positive rate (TPR).

In general, runtime grows as m increases and as β *decreases*. The reasons for the former are that as more simulations are required to check whether they are compatible with O the more time takes to perform the computation. However, the relation between run–time and β is not that obvious. We note that as β decreases, the decision making gets more random, since differences between $Q(b, a)$ values tend to be ignored. This usually leads to significantly longer executions.

We will next take a look at the OFFICE domain and compare the recognition accuracy of the quasi–greedy Boltzmann policy resulting from setting $\beta = 40$ with the greedy policy as the number of samples taken m increases. One would expect to obtain perfect recognition when using the same policy that generated O, but we will see that this expectation ignores the possible – and frequent – ambiguity of actions in O.

Figure 5.2 shows the accuracy of using both policies to obtain our estimates of $P(O|G)$ over observation sequences with very few actions on the OFFICE domain. First, we can see that the GR$^{m,\beta}$using the greedy policy is not achieving maximum accuracy. Second, and startlingly, the quasi–greedy policy is performing slightly better than the greedy policy. We offer next a cogent explanation for both phenomena.

The reason for not achieving maximum accuracy is that for O such as

$$\text{walk}(corridor, lab),\ \text{walk}(lab, corridor),\ \text{walk}(corridor, club)$$

$P(O|G)$ is equal for all three goals. While the joint goal is discarded because of having a lower $P(G)$, there is no information available to identify the true hidden goal. Therefore, there is a minimum attainable FPR which depends on whether policies for goals share some action subsequence and the observation level, so the lower it is the *more likely* is to obtain a sequence which perfectly fits both goals. This issue has a greater effect in the KITCHEN and DRAWERS domains.

The value functions $V_G(b)$ computed by RTDP–BEL are optimal in the sense that they minimize the *expected* cost to achieve G, which can be a quite conservative estimate of the actual cost of the resulting execution. The greedy policy for the goal read() of the OFFICE domain will take the agent directly to the cabinet in her office to fetch blank paper *just in case* the printer is out of paper. However, with

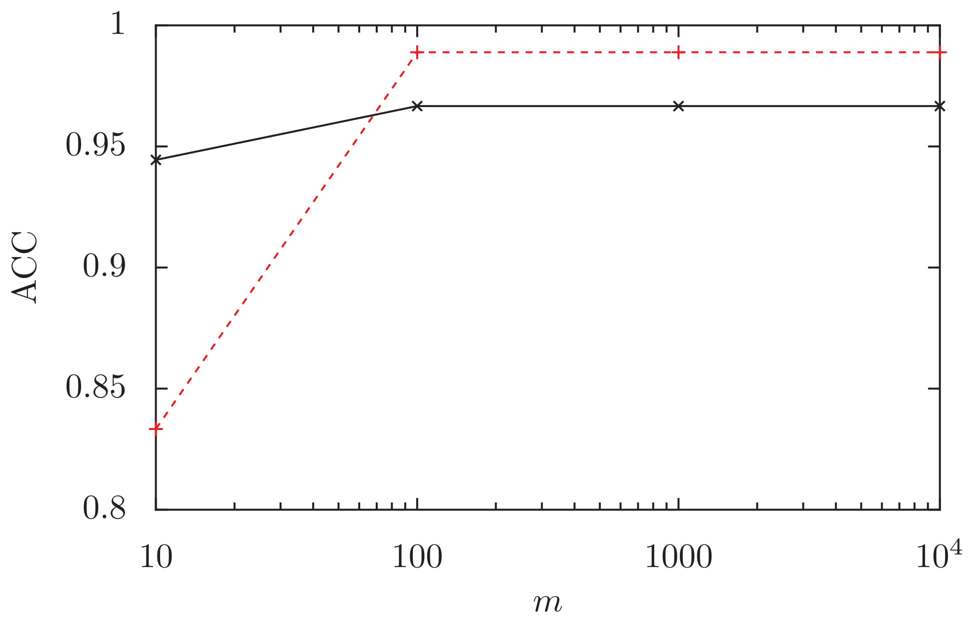

Figure 5.2: Goal recognition accuracy of the Boltzmann policy setting $\beta = 40$ – dashed line – and the greedy policy – continuous line – on the OFFICE domain over increasing values of m. Observation sequences tested were obtained from sampling the 30% of actions featured by greedy policy executions.

the Boltzmann policy there is a chance that the agent will get nowhere near to the cabinet while pursuing the *print article* goal.

The agent might not take the action a that minimizes $V_G(b')$ for b' resulting from execution a *and* the printer might have paper available so there is no need of blank paper. On the other hand, the agent will necessarily get to the cabinet to achieve the *have coffee* goal, since she needs to take the cup inside. When trying to recognize the hidden goal for an observation sequence O such as

$$\text{walk}(\textit{corridor}, \textit{lab}),\ \text{walkTo}(\textit{cabinet}), \text{walk}(\textit{lab}, \textit{corridor})$$

with the greedy policy, both goals have the same posterior probability. In contrast, when $\text{GR}^{m,\beta}$ is using the Boltzmann policy with $\beta = 40$, the drank() goal is slightly more probable than read(). For $m = 1000$, as many as 100 out of the 1000 simulations involved did not require the agent to fetch paper to achieve *print article*, lowering the likelihood of that goal. As in the example above, the classifier $\text{GR}^{m,\beta}$ using the greedy policy incur in a false positive error, while the classifier using a quasi–greedy policy does not.

5.8 Summary

This chapter has extends the approach to model–based, generative plan recognition introduced in Chapter 4 to consider models where the outcomes of actions done by the agent are not deterministic and the feedback provided to the agent by the environment is limited. The computation of $P(O|G)$ now consists in sampling executions of a Boltzmann policy defined over the value function V^* associated with the Goal

MDP or Goal POMDP domain that models agent behavior. The value function V^* is computed off–line by the GPT planner, which is not modified in any way.

Several domains are discussed in depth, describing their actions and states, showing the flexibility granted by the factored modeling language supported by GPT. Experimental results over these domains are presented, and show the approach to be robust and to provide with accurate estimates of the posterior goal distribution $P(G|O)$ for these domains.

Part III

Discussion

Chapter 6

Approaches to Plan Recognition

In this chapter we review some of the most significant works on plan recognition produced over the past thirty years. The content of the present chapter is not meant to be a complete and thorough survey of the field, but rather a selection of those approaches the author finds most relevant to provide a context for the model–based framework for plan recognition introduced in Chapters 4 and 5.

6.1 BELIEVER

The earliest, plan recognition system proposed was Believer (Schmidt et al., 1978). Portrayed as the implementation of a psychological theory of how human observers understand the actions of others, the authors aimed at offering a comprehensive framework to account for three different queries related to observed actions:

1. Recognizing the goal of the observed agent, which is referred in the paper as "summarizing observed agent actions".
2. Inferring missing actions from the input sequence, referred as "recalling actions".
3. Predicting the actions the agent will do after the observed sequence.

Believer inputs are two: the observed action sequence, and a model of agent behavior and its environment. This model is an extension of the original Strips (Fikes and Nilsson, 1971) planning model, that allows to describe agent beliefs on the actual world situation and non–deterministic action effects, thus closely matching the contemporary definition of *contingent planning* (Bertoli et al., 2001) problems.

The authors break down their planning domain into three separate sub–models: the *Person* domain, the *World* domain and the *Planning* domain. The *World* domain consists of statements, made in the language of first–order logic, that the observer believes to be true about the world, such as if a particular object is stored in a container or not. The *Person* domain specifies the beliefs and desires of the agent on particular statements in the *World* domain by introducing second–order statements such as *Beliefs*, *Knows*, *Likes* and *Wants*. Finally, the *Planning* domain is composed of action *schemas* and the definition of *valid* plans. Such schemas are parametrized descriptions of the actions the agent can do, the conditions that enable them and a separate specification of the *actual* outcome of the action and the *intended* outcome of the action. Table 6.1 depicts the action schema describing the act of moving between two different places.

Field	Value	
Name	Walk	
Parameters	Name	Type
	Actor	Person
	From	Location
	To	Location
Opportunities	$(Loc\,Actor\,From)$	
Outcome	$(Loc\,Actor\,To)$	
Goal	$(Loc\,Actor\,To)$	

Table 6.1: Schema for the Walk action discussed in (Schmidt et al., 1978). Opportunities refer to action *preconditions*, Outcome is the actual action effect and Goal is the intended action effect. In this particular example, action intended and actual outcomes are always the same.

It is interesting to note that the definition of valid plans is left as something particular to the plan recognition problem at hand. In the light of current practice in planning such an statement results baffling, and certainly, of little practical value. In fact, the authors are referring to the causal structure implicitly available on action schemas, or possible causal links (Sacerdoti, 1975) between actions in a valid plan. Input observation sequences are composed of grounded action schemas, along with a precise description of what statements in the World and Person domains become true and false, and finally, a complete description of agent beliefs and hypothetical goal. In other words, (Schmidt et al., 1978) notion of O conveys much more information than required in our definition of observation sequences O given on Section 4.1, which only contains grounded action schemas. Believer is not really deducing what hypothetical goals explain best observations, but rather checking whether, under the assumptions implicit in the models given for the agent and its environment, some certain goal is a logical consequence of observed actions.

At its core Believer consists of a heavily modified version of Sacerdoti's planner Noah (Sacerdoti, 1975), which is used to generate plans for the hypothetic goal encoded in the input observation sequence. Once the plan is generated, it is checked whether it is consistent with the input observation sequence. If it is not, then a new plan is generated taking into account the conflicts between the previously generated plan and the input observation sequence. This so–called "revision process" is implemented by mean of plugging new *plan critics* – that is, pruning rules – to the planner backward–chaining search process which are selected from a hand–coded library of pruning rules specific for the input planning domain.

Believer prefigures many aspects of subsequent approaches to plan recognition. It proposes a very rich language for reasoning on plans based on Strips, yet heavily relies on knowledge engineering both in the characterization of agent behavior and also in the process of searching for plans that account for the input sequence. Surprisingly, the planning domains discussed by Believer authors consider single statements – rather than a conjunction of statements – as agent goals. The reasons for them not considering slightly more complex, but much more expressive, goal descriptions are not mentioned at all, though the action schemas discussed actually contain preconditions consisting of conjunctions of fluents. Believer is also the first system where it is discussed the implications of assuming the observed agent to be cooperative, non–cooperative, or just not caring at all about being observed.

As with much work done on Artificial Intelligence in the late 70s, Believer is more the *concept* for a plan recognition system that an actual, practical system. There are lengthy text descriptions of examples illustrating the plan recognition process, but there is no sign of Believer *actually* solving any plan recognition problem.

6.2 Plan Recognition and Natural Language

The next approach to plan recognition we will present was not actually meant to address plan recognition but rather, an application of plan recognition, that of inferring the purpose questions made by a speaker (Cohen et al., 1981). The authors are clearly inspired by the work of Appelt (Appelt and Luria, 1982), who took automated planning modeling languages (Fikes and Nilsson, 1971) and solvers (Sacerdoti, 1975) off–the–shelf and applied them to solve the problem of natural language generation [1].

While similar to the approach taken in Believer (Schmidt et al., 1978), Allen and Perrault (Perrault and Allen, 1980) approach is less ambitious and at the same time, way more clear and appealing. The relevant automated planning concepts and algorithms are described in a precise manner and concrete computational results are sought.

Several simplifying assumptions are made by (Perrault and Allen, 1980) which are commonly made in subsequent research on plan recognition:

1. Plan recognition involves two entities, the *observer* and the *observed* agent, and the observer is assumed to share the planning domain with the observed agent.

2. It is assumed that the observed agent is not changing goals while it is being observed.

3. The observed agent is cooperative, that is, she wants the observer to *understand* his actions, in an intentional or does not want to hide her intentions from others.

These assumptions allow the authors to consider the processes of plan generation, done by the observed agent, and that of goal inference, done by the observing agent, as one single chain of plausible inferences operating on goals and observed actions. This is achieved by computing plans that contain, simultaneously, actions consistent with the observations made and actions required to achieve one of the hypothetical goals. Again, goals are not considered to be a separate entity from plans, but rather, a mere sub–product of the computed plan.

[1] Research on natural language generation continues to rely on developments in automated planning to this day, see (Koller and Stone, 2007) and (Rieser and Lemon, 2011).

The planning algorithm presented by Perrault & Allen actually corresponds to Pocl (Weld, 1994), that is, planning in the space of partially ordered plans, rather than the state–space search methods discussed in Chapter 2. Branching on the space of *partial* plans is guided by the following heuristics and pruning rules:

1. Select actions so to make true the preconditions of actions *already* in the plan but still false.
2. Prune partial plans where one or more of the observed actions cannot be included in the plan.
3. Select actions whose effects are relevant to the goal being sought by the observed agent.

The third heuristic is actually a domain–specific hard–coded decision making procedure, and the authors do not give any further specific details. Whenever for a given goal no complete plan can be found, the goal is dismissed as a valid or relevant goal for the partially observed plan in the input.

Allen and Perrault work is of special significance to this thesis. As we will present in Chapter 4, we make very much the same assumptions made in (Perrault and Allen, 1980) and we seek the same objects: plans that comply with the goals and the observations simultaneously. Nonetheless, we achieve this in a more principled way and need not to modify the planner by introducing hard–coded heuristics. Actually, heuristics 1 and 3 above are the foundation of the domain–independent heuristics computed automatically by contemporary planners.

6.3 Plan Recognition as Minimum Set Cover

One of the works with the longest term influence into plan recognition research is that by Henry Kautz and James Allen, first presented in (Kautz and Allen, 1986b) and recapitulated a few years later in (Kautz, 1991). Kautz's framework makes two major contributions, one methodological and the other technical. The first contribution is that it clearly specifies what conclusions – goals – are absolutely justified on the basis of the observed actions, the observer's knowledge about the observed agent possible behaviors and a number of explicitly made "closed world" assumptions. This contrasts starkly with the vague definitions one can typically find in (Schmidt et al., 1978) or (Perrault and Allen, 1980). The second contribution is that for the first time the computation of the conclusions drawn on the observed agent behavior does not rely on ad–hoc heuristics to guide the inferences made by the plan recognition system, but are rather suggested by the theory being proposed.

Rather than representing the observer knowledge with a theory that implicitly describes possible observed agent behavior, Kautz representation is an explicit description of possible behaviors in terms of an *event hierarchy*. Events can be either actions or whole plans, and the hierarchy, besides accounting for inclusion of actions in plans, also allows to express functional relationships between actions, such as that of precedence.

Figure 6.1 depicts a simple event hierarchy presented in (Kautz, 1991). At the root of the hierarchy we find the *End* event, which accounts for the observed agent

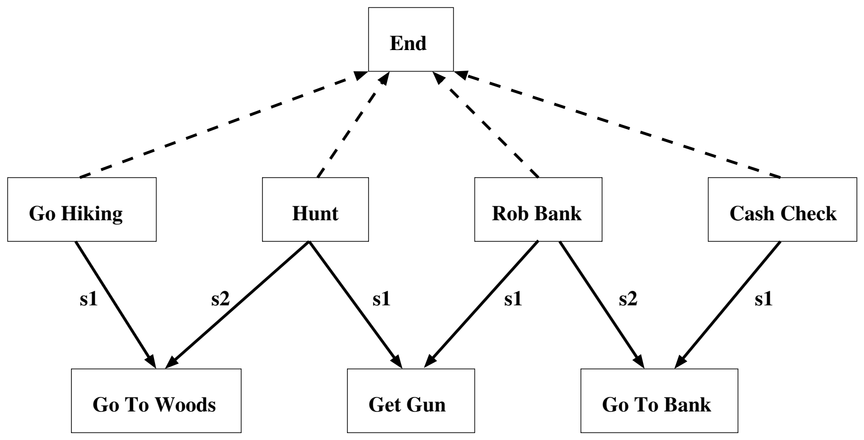

Figure 6.1: Example event hierarchy in (Kautz, 1991). Boxes denote events, dashed arrows represent specialization – *is a* – relationships and solid arrows represent inclusion – *has part* – relationships. Inclusion relationships between events are indexed according to their order e.g. s_i tokens denote that event comes before any other labeled as s_j, $j > i$. Plans encoded by this hierarchy are discussed in the text.

achieving a goal. Specialization relationships allow to specify what are the set of possible plans the agent will be executing, or alternatively, the set of possible goals the agent might be pursuing. Inclusion relationships allow to express how a certain plan can be decomposed into a totally–ordered sequence of sub–plans or primitive actions. The hierarchy in Figure 6.1 is encoding a total of 4 different plans, each plan corresponding to a single goal:

- For the goal *Go Hiking*, the plan is ($GoToWoods$).
- For the goal *Hunt*, the plan is ($GetGun$, $GoToWoods$).
- For the goal *Rob Bank*, the plan is ($GetGun$, $GoToBank$).
- For the goal *Cash Check*, the plan is ($GoToBank$).

While the notion of event hierarchies naturally appeals to a graphical representation, it is certainly unwieldy to define what are its *semantics*: the set of possible plans. Kautz actually encodes event hierarchies as a knowledge base made up of first–order axioms which describe precisely the plans. The event hierarchy in Figure 6.1 is thus encoded as the following set of first–order axioms:

1. $\forall x. GoHiking(x) \supset End(x)$
2. $\forall x. Hunt(x) \supset End(x)$
3. $\forall x. RobBank(x) \supset End(x)$
4. $\forall x. CashCheck(x) \supset End(x)$
5. $\forall x. GoHiking(x) \supset GoToWoods(s1(x))$
6. $\forall x. Hunt(x) \supset GetGun(s1(x)) \land GoToWoods(s2(x))$
7. $\forall x. RobBank(x) \supset GetGun(s1(x)) \land GoToBank(s2(x))$

8. $\forall x. \mathit{CashCheck}(x) \supset \mathit{GoToBank}(s1(x))$

as well as further axioms encoding *assumptions* needed so that the event hierarchy is *complete*, that is, whenever an event different from *End* occurs, it must be part of some other event, and the relationship from event to component appears in the event hierarchy. The assumptions are:

1. *Exhaustiveness* – Or closed world assumption: the ways of specializing an event type in the event hierarchy are the only ways of specializing it. In other words, there are no valid plans other than the ones encoded in the models of the knowledge base.
2. *Disjointness* – The different ways to specialize the *End* event are mutually exclusive. That is, the observed agent is assumed to be pursuing *exactly one* of the "top–level" goals.
3. *Use* – A plan or action implies the disjunction of the plans which use it as a component.
4. *Minimum Cardinality* – The number of plans and actions is minimized.

Making these assumptions allows Kautz to formulate a model theory for plan recognition, by borrowing the model theory of circumscription (McCarthy, 1980).

The plan recognition process then, consists in deducing from E, the event hierarchy and O, a first–order formula encoding what actions (events) are observed and in what order, a minimum–cardinality truth assignment for the predicates in E. The predicates specializing the *End* predicate which are true in such an assignment, are then the minimum set of possible goals that justify the observations.

This computation, in the general case, is undecidable and both (Kautz, 1991) and posterior works relying on the notion of event hierarchy or plan libraries have stuck to a purely propositional representation, which while still intractable – minimum cost satisfiability is NP–HARD – it can be effectively approximated in a greedy manner.

The methodological and technical contributions discussed above come at a great price, as recognized in (Kautz, 1991). The above is only appropriate for domains where one can enumerate in advance all plans achieving a goal, or those domains where one is interested in recognizing stereotypical behavior, that is, only a small fraction of the possible plans is deemed interesting. The assumption of complete knowledge on part of the system designer is *fundamental* for the validity of the approach.

This is admitted by (Kautz, 1991) to be a case of "making virtue out of necessity". The flexibility of previous approaches (Perrault and Allen, 1980), where the space of possible plans is infinite, entailed a search problem which was well out of reach for the planners available at the end of the 1980s. Quoting (Kautz, 1991)

> We decided to maintain the assumption of complete knowledge, and only construct plans by specialization and decomposition, as described below, *until we have developed methods of controlling the combinatorial problem.*

This observation was all but forgotten, as plan recognition research eventually became out of touch with the planning community. So from the early 90s to the present thesis, almost nobody – the one notable exception is (Lesh and Etzioni, 1995) – made again the connection with automated planning, even when significant advances were made in the early nineties, with the advent of GraphPlan (Blum and Furst, 1995) and SatPlan (Kautz and Selman, 1992).

Another unfortunate consequence of compiling possible plans into event hierarchies or plan libraries was that of blurring a previously crisply defined concept, that of *goal*. In approaches based on automated planning models, goals are Strips goals: formulas specifying a particular truth assignment on predicates modeling world state. Posterior research on plan recognition missed the correspondence of (Kautz, 1991) *End* root event, with that of the *End* action in planning systems such as (Currie and Tate, 1991) whose precondition is the goal formula. This is not surprising since event hierarchies and plan libraries *compile away* world states, throwing away significant information. Indeed, if all possible plans are those encoded in the hierarchy, then it does not matter what was the particular world state supporting actions and plan execution.

The most charring criticism of (Kautz, 1991) is that set minimization, as the computational framework for an abductive reasoning task such as plan recognition is not adequate (Charniak and Goldman, 1993). In Kautz's formulation, a goal G consisting of the conjunction for goals G_1 and G_3 from the example described in Section 1.2, would be less likely than either G_1 or G_3 regardless of the observation sequence O. In other words, it is assuming that the agent is more likely to pursue "simpler" goals, or if we used our planning–based formulation in Chapter 4, the goals with *cheaper* optimal plans. Something that might well be a common occurrence in many settings, but which is ways better to make this assumption explicitly in a Bayesian framework by setting priors $P(G)$ accordingly, than to hide it inside the computational machinery of the formulation.

Therefore, after (Kautz, 1991), plan recognition research veered towards *reasoning over knowledge bases* rather than *reasoning over plans*, as the focus and the efforts of the plan recognition community was put on *matching* a pattern – the observed action – against a database of plans rather than on *deducing* the purpose of the pattern.

6.4 Probabilistic Models for Plan Recognition

(Charniak and Goldman, 1993) argue that the problem of plan recognition is a problem of abductive reasoning under uncertainty, and propose the framework of Bayesian probability theory and Bayesian networks (Pearl, 1988) for inferring explanations of the observed sequences of actions.

Plan recognition is then formulated as the problem of building and evaluating *plan recognition networks* or Prn. These Bayesian networks are to be derived according to a set of provided *construction rules* which map knowledge about actions and plans into variables and conditional probability tables, resulting in factored representations of posterior probability distributions $P(\pi|O)$ over some set of *hypothetic* plans π. The authors acknowledge that the construction procedure can indeed lead to infinite size networks for very simple planning domains.

Knowledge about plans and actions is represented a first–order logic program, where the head of the rule denotes a *hypothetic plan* and the body is a conjunction of clauses encoding constraints that prescribe which are the possible actions – or sub–plans – that can be components of the hypothetic plan. Plans are then encoded as a conjunction of n clauses of the form:

$$(inst\ p\ \text{plantype})(=(stp_1\ p)a_1)\ \ldots(=(stp_n\ p)\ a_n)$$

where each $(=(stp_i\ p)a_i)$ clause denotes that the action a_i is to be executed on the plan step or slot stp_i. The *stp* function sub–index, though, does not have temporal semantics denoting just membership. In order to infer the plan – if such plan exists – from observed actions, it is necessary to define what actions a_i can fill plan steps stp_i. It can be seen that (Charniak and Goldman, 1993) representation of planning domains is not very different from the plan libraries used in (Kautz, 1991).

These logic programs are then encoded into a Bayesian network N by mapping each proposition in the knowledge–base into boolean variables. Hypothetic, top–level, plans are root variable nodes ($inst\ p$ plantype) of the network. These are linked to $(=\ (stp_i)\ a)$ nodes, which in turn are linked to the nodes representing the actions or plans a that can fill up the step (stp_i) as defined. Links between nodes are fleshed out in a straightforward manner. Hypothetic plans prior probability distributions $P(\pi)$ and conditional probability distributions $P(a|stp_i)$ are assumed to be Bernoulli distributions, unless otherwise noted. The networks resulting from this mapping are not necessarily acyclic, since slot clauses might be related by more than one plan, and a plan might be a sub–plan of other plans.

This translation procedure runs into problems because an action a can well be a valid action for a slot in many plans, and because of allowing plans to be recursively defined in terms of actions and sub–plans, leading to a network structure that would be equivalent to a clique with all nodes connected or to infinite size networks. Analogous problems to the two above can be found on (Kautz, 1991) proposed plan libraries.

The first problem is addressed by first, computing paths between actions and hypothetic plans in the moral graph of the Prn implicitly described in the plan and action knowledge base. Then these paths are evaluated by a set of domain–specific rules that prune away "meaningless" paths. The problem posed by recursive plans is not discussed any further after being introduced.

No details are either given on the computational complexity of evaluating the resulting networks. However, given the small size of the examples presented in the text, the subsequent Map query posed to the network seem to be well within the limited computational capabilities of the time. The computational intractability of Map Bayesian networks queries, and how to overcome that problem by relying on approximated algorithms, has been a major topic in most work that followed Charniak and Goldman's notion of plan recognition, such as (Bui, 2003), which will be reviewed in Section 6.6.

The present thesis owes to (Charniak and Goldman, 1993) the insight of considering plan recognition as a probabilistic abductive reasoning task. However, the formulation presented on Chapter 4 represents plans and actions in a planning formalism and posterior goal probabilities are deduced from the properties of solutions to planning models, computed with dedicated planning solvers. While the issue with recursive plans does not apply at all to our formulation, the issue regarding an action – or sequence of actions – being present in optimal plans for many goals G does apply to some extent.

This issue is addressed by the way we approximate likelihoods $P(O|G)$ presented on Section 4.7 implicitly addresses this, since it builds the estimates $P(O|G)$ by selecting just one amongst all plans for a goal G with the same cost, and actions in those plans are selected in order to minimize that cost. Otherwise than that, having actions which are relevant to many goals does not pose other problem than that of very short observation sequences O leading to many goals being assigned the same $P(G|O)$, which is hardly an issue that can be addressed.

6.5 Planners Generating Plan Libraries

The work by Lesh and Etzioni (Lesh and Etzioni, 1995) is a model–based plan recognition formulation that is at first sight quite similar to the one presented in Chapter 4. First, rather than representing the possible agent plans explicitly in a plan library, it also uses a planning domain expressed in a language closely related to STRIPS. Second, solutions to plan recognition problems are defined as subsets of a set of hypothetic goals, similarly as we do in the qualitative plan recognition account described on Section 4.5. However, as we discuss next, there are some significant and critical differences.

The first and most obvious difference lies in the observation sequences O considered by Lesh & Etzioni and the observation sequences we consider in Section 4.1. While the former are required to be *plan prefixes*, that is the first n actions in some plan, the latter might be any subset of actions in a plan. Hence, Lesh & Etzioni's formulation of plan recognition problems is a special case of the more general formulation that we have presented.

Lesh & Etzioni – as we do – compute plans that satisfy the observation sequence O. However, while we consider *cost optimal* plans, Lesh & Etzioni formulation relies on the notion of *irredundant* valid plans. Irredundant in this context means that plans are guaranteed to feature only actions a which either add one of the facts in the goal G, or a fact which is in some precondition of an action in the plan.

This notion is weaker than that of optimal plans, since all optimal plans are irredundant, while there are potentially many irredundant plans which are not optimal. Therefore, the semantics of the goal subset computed by Lesh & Etzioni are very different from that of the *optimal goal sets* introduced in Definition 4.5. Optimal goal sets contain those hypothetic goals whose best plans satisfy observations, capturing in a limited way the degree in which those observations are *necessary* in order to achieve the goals. In Lesh & Etzioni formulation is not taking into account that irredundant plans might achieve several goals as a *side effect*, and therefore the presence or not of a goal in the selected goal set, does not give a good leverage on the observed agent intentions.

The computational framework discussed in (Lesh and Etzioni, 1995) relies on having compactly encoded the set of *all* possible plans the agent can execute in a directed graph, with nodes corresponding to actions and goal formulas. Edges connect two nodes n, n' if n is an action adding facts in the precondition of the action or the goal formula represented by node n'. Lesh & Etzioni assume this graph to be computed in an off–line manner. However, as the authors acknowledge, such a graph can easily become unfeasible to compute or store as its size grows exponentially with the length of the longest plan considered.

Irredundant goal sets are obtained by applying iteratively a set of pruning rules which discard nodes and edges when there is no path between the nodes representing the actions in the observed plan prefix. While Lesh & Etzioni claim the described procedure to be sound, that is, limited to consider irredundant plans, it is not proved.

While in some respects this work is quite close to the formulation of plan recognition we have presented, in other aspects follows the practice of performing its computation on a explicit representation of possible plans, and amounts to be a plan–library approach to plan recognition with the novelty of libraries being computed from planning domains by doing a potentially very big number of calls to a planner.

6.6 Plan Recognition and Dynamic Bayesian Nets

The work comprised by (Bui, 2003) and (Bui et al., 2008) is a quite clear and coherent integration of two key lines of research pursued in plan recognition. One is that of deeming plan recognition as a problem of probabilistic inference over a Bayesian network encoding possible agent plans, initiated by (Charniak and Goldman, 1993) and already discussed in Section 6.4. The other is that of considering agent actions to have non–deterministic effects, an extension to the classical planning model implicit in (Kautz, 1991) characterization of agent actions and plans and first discussed by (Pynadath and Wellman, 2000).

Bui's framework for plan recognition departs from (Charniak and Goldman, 1993) in two key aspects. First, rather than having a dual representation of plans and actions, it does away with the logic program representation of the plan library, and models directly agent behavior as a Bayesian network. Another crucial difference is that Bui's capitalizes Dynamic Bayesian network (DBN) development (Murphy, 2002) in order to account for the passage of time. DBN's allow to model temporal data – such as the sequence of observed actions of interest in plan recognition – in a natural way, so that conditional probabilities in the nodes corresponding to events occurring at different points of time can be estimated independently.

As in (Charniak and Goldman, 1993) the networks used by Bui to model agent behavior are not amenable to exact inference. This difficulty is bridged relying on the well–founded and efficient Rao–Blackwellised particle filter proposed by (Doucet et al., 2000) to perform approximate probabilistic inference over DBNs.

A more interesting departure in the formulation lies in modeling agent behavior as a Mdp. As discussed in Section 3.1, when actions have non–deterministic effects, the results of such action can no longer be predicted with full certainty. This kind of agent models render plan libraries useless, if effects of actions can no longer be predicted, subsequent actions cannot be either. Such a setting was first considered by (Pynadath and Wellman, 2000) that proposed a representation of agent plans and actions based on probabilistic grammars. Bui takes this idea a step further by introducing a control automaton producing observed actions and encoding it into

Hierarchical Hidden–Markov Models (HHMMs) (Murphy, 2002), a specific type of dynamic Bayesian networks.

This an efficient and crisp formulation of plan recognition, which suffers heavily from the lack of expressive power to capture succinctly agent behavior. HHMMs handle very well tasks which can be easily mapped into the problem of traversing a graph. However, since no factored planning language is used, it relies in representing such graphs explicitly, something that is hardly feasible unless very simple domains are considered. Furthermore, it can not handle at all interleaved execution of plans. In this respect, Bui's approach inherits the problem suffered by plan–library approaches, where something simple such as considering goals – or top–level plans – which are a combination of other goals becomes extremely complex or just unfeasible.

Yet another problem is that of handling incomplete observation sequences. In this case the problem does not lie on actions being missing, but in determining which observation node in the HHMM corresponds to the first action in the observation sequence.

6.7 Plan Recognition as Parsing

Perhaps the approach to plan recognition that has generated the most literature is that of formulating plan recognition tasks as *parsing* (Vilain, 1990). In this approach possible agent behavior – in the form of plans with deterministic action effects – is encoded into a grammar Γ, where terminal symbols represent actions and non–terminals with multiple production rules associated represent *tasks*. The start symbol of Γ represents the goal G that the agent might want to achieve. Observation sequences O are then checked to be terminals produced by grammar Γ.

In order to decide if the grammar Γ encoding the plans for G is a good explanation for O, it is checked – with parsing algorithms – whether Γ can produce the sequence of terminal symbols (actions) in O.

Non–terminal symbols in Γ are representing goals and sub–goals, along with the set of alternative plans – the set of possible of productions of the non–terminal – that can achieve them. This representation is remarkably similar to the notion of *tasks*, which can be found in the independently developed notion of Hierarchical–Task Networks (HTN) (Erol et al., 1994). HTNs are regular STRIPS planning domains augmented with *methods* that implicitly describe a set of plans that achieve the goal represented by the task.

However, grammars and task networks are far from being equivalent. HTNs take into account explicitly the notion of *world state* as represented by STRIPS states defined in Section 2.2. Grammars Γ, very much like plan libraries, do not represent states explicitly. Like libraries, states are implicitly represented in the *context* of an action, the path that goes from a node representing a top–level goal to that particular action. In the case of plan recognition as parsing formulations, these are paths in the *parse tree* of the grammar.

PHATT (Geib and Goldman, 2009) is a recent system based on the parsing approach to plan recognition which subsumes in an elegant way past formulations and provides with posterior goal distributions $P(G|O)$ over a set of hypothetical top–level goals G.

Rather than obtaining $P(G|O)$ formulating plan recognition as a problem of probabilistic inference solved by using some form of belief propagation over a Bayesian network as in (Charniak and Goldman, 1993; Bui, 2003), PHATT uses the *generative* capabilities of grammars Γ to obtain an approximation of $P(G|O)$. This may appear somewhat similar to probabilistic formulation of plan recognition as planning we proposed in Section 4.7.

There are however several notable and key differences. While the basis for our probabilistic model are costs of plans which only depend on the definition of the planning problem $P[G]$, (Geib and Goldman, 2009) requires to augment the grammar Γ with probability distributions for production rules. The purpose of these is to model the *preferences* the agent might have to accomplish a certain goal or sub–goal with one particular plan. Yet another difference can be found in how $P(G|O)$ is defined

$$P(G|O) = \sum_{e} P(G|e)P(e|O)$$

where e are the possible *explanations* in Γ for O. These explanations consist of a pair of partial derivation trees for grammar Γ where each action in O corresponds to a terminal symbol, along with the set of *substitutions* of non–terminal symbols corresponding to each partial derivation tree.

$P(G|e)$ is defined to be 0 for those explanations that do not include the top–level non–terminal accounting for the goal G, and 1 if they do. The likelihood of a plan given a sequence of observations $P(e|O)$ is

$$P(e|O) = \alpha P(e) \cdot P(O|e)$$

where $P(O|\pi)$ is the probability for e to *generate* O and α is a normalization constant. This should not be confused with our definition of $P(O|G)$ in Equation 4.2

$$P(O|G) = \sum_{\pi} P(O|\pi) \cdot P(\pi|G)$$

since explanations e are not *single* plans π. Explanations are actually the *sets* of plans that achieve O and satisfy O as per Definition 4.2. While the approximation of $P(O|G)$ we propose in Section 4.7 is obtained from one of the *best* possible plans that satisfy O, PHATT computes $P(O|e)$ – distribution which has the same semantics as $P(O|G)$ – considering *all* plans in the library that satisfy O.

In order to obtain $P(O|e)$ PHATT relies on a substantial number of parameters consisting of three different types of probability distributions that characterize the chances of a certain production rule in Γ to be activated. Namely, the prior probability of goals, the probability of choosing a grammar rule amongst other rules associated to the same non–terminal and the probability of choosing to expand one non–terminal over another, when two of them appear unordered in a production rule body. This contrasts with the reduced number of parameters in our formulation, costs of actions a and the prior probability distribution over hypothetical goals $\mathcal{G}_T$.

$P(O|e)$ is computed in two steps. First, the probability distributions relevant to the explanation are obtained by traversing the partial derivation trees in the explanation.

Second, $P(O|e)$ is directly computed as the product of the retrieved probabilities for each derivation step. While the computation procedure described in (Geib and Goldman, 2009) is straightforward, its computational complexity is **NP**–hard, the same as that of searching for plans solving a Strips planning problem $P[G]$. Furthermore, we find that there are some implicit assumptions being made that consider the probability for selecting a production rule and that of the order of expanding non–terminals to be independent.

6.8 Plan Recognition as Inverse Goal MDP planning

The last work we will review in this chapter (Baker et al., 2009) is another model–based approach to plan recognition which computes posterior goal distributions $P(G|O)$. This paper is a noteworthy example of the interest that plan recognition raises in fields other than Computer Science, such as the Cognitive Sciences.

We find this work to be of particular relevance to this Thesis, since we find it to be the approach which is the closest, conceptually and methodologically, to the contributions discussed in Chapter 4 and Chapter 5. Plan and goal recognition models and algorithms to solve these are of special relevance to the Cognitive Sciences studies on the nature of *anticipatory systems* (Rosen, 1985):

> A system containing a predictive model of itself and/or its environment, which allows it to change state at an instant in accord with the model's predictions pertaining to a latter instant.

Cognitive agents are conceived as such systems, able to reason about the future and fulfill their goals relying on these predictive models. Experimental evidence of the existence of such anticipatory mechanisms in the human brain has been gathered in recent years (Pezzulo, 2008) and it has been observed to be of crucial importance in a number of cognitive functionalities such as vision, motor control, learning, motivational and emotional dynamics. The availability of robust, efficient and workable computational models is crucial for Cognitive Sciences because they allow to test hypothesis about such representations in a natural way by contrasting the observed behavior of simulated agents with that of actual humans.

In contrast with other formulations of plan recognition originated outside of the AI community, such as (Schmidt et al., 1978) Believer, (Baker et al., 2009) relies on a crisply defined mathematical formulation and a solid and well–defined planning model, that of Goal Mdps described in Chapter 3.

(Baker et al., 2009) model the agent executing a Boltzmann *policy* π for a goal G, where the probability of the agent executing an action a on a given state s is defined as

$$P(a|s) \propto \exp\{\beta Q_G(s,a)\}$$

which approximates the *principle of rationality* (Dennett, 1983) discussed in Section 4.2 and $Q_G(s,a)$ is the expected cost to reach a goal state S_G from the current state s. We have directly borrowed this idea and used it in our formulations of plan recognition in Section 4.7 and Section 5.2. We note that this is not an essential component of our formulation nor that of (Baker et al., 2009), both can be trivially parametrized over the action selection probability distribution.

$Q_G(s,a)$ values are assumed to be defined over the optimal value function V^* of the Goal MDP $M[G]$, which is computed off–line using the Dynamic Programming algorithms discussed in Section 3.2. We note that the classical planning problem state models $S(P)$ are a *special case* of Goal MDPs, since these can be readily obtained from $S(P)$ by replacing transition function $f(a,s)$ by probability distribution $P(s'|s,a)$ such that

$$P(f(a,s)|s,a) = 1$$

and set to 0 for any state $s' \neq f(a,s)$. For this special case, our formulation in Section 4.7 rather than computing $V^*(s)$ for all s, it is computing these on a *as needed* basis using a classical planner.

The biggest difference between our formulation and that of (Baker et al., 2009) lie at the definition of observation sequences O and in the flexibility our formulations enjoy because of using a factored representation of both classical planning problems and Goal MDPs. Regarding the former, (Baker et al., 2009) only consider observations O which are *prefixes*, while we allow O to be any subsequence of the actions appearing on possible plans or selected when executing a policy. This allows our formulations to be useful in a greater variety of plan recognition tasks.

The fact that (Baker et al., 2009) does not use a factored representation forces them to put forward *three* different formulations of plan recognition where increasingly less restrictive assumptions on the behavior of the observed agent are made. The most simple formulation assumes that the agent is seeking one single goal G, that is, reaching a state S_G. While this might seem equivalent to our definition it is not. Since we use a factored representation of states s, our formulations can define *implicitly* the set of goal states S_G and combine goals to consider the conjunction or the disjunction of two goals G an G'. This simple operation becomes easily unwieldy when dealing with a explicit representation of the planning task.

The second formulation caters for agents that change goals over time. From a planning perspective this is somewhat ill–defined, since the agent is actually expected by (Baker et al., 2009) to switch goals *randomly* according to some parameter γ. We find this noisy goal switching notion to attempt to capture plan recognition tasks where achieving a goal G involves reaching some states s in no particular order. We observe that such random goal switching is at odds with the principle of rationality. If the agent is assumed to prefer engaging in optimal behavior, then the order in which these goals are achieved will probably matter. The third an most general formulation covers the case of agents whose goal requires visiting some to visit some states s' before reaching a goal state in S_G, in some particular order.

The three formulations proposed by (Baker et al., 2009) are subsumed by ours, since they can be accounted either by composing goals or tweaking the definition of $P[\cdot]$. We note that when a goal G is defined as the conjunction of two other goals G_1 and G_2, the order in which those goals are achieved is left for the agent to decide. So rather than having an agent that randomly switches between them, in our case the agent is actually computing the order – and possibly interleaving the pursuit of either goal – in a way that its performance is most efficient.

Accounting for the third model involves changing the definition of $P[\cdot]$ so that the goal G can only be achieved if plans that achieve G_1, ..., G_n in this order. This can be done by introducing dummy fluents p_{G_i}, p_G and adding conditional effects

$$G_i \wedge p_{G_{i-1}} \rightarrow p_{G_i}$$

in the definitions of actions a such that $G_i \cap Pre(a) \neq \emptyset$ [2], as well as a dummy action with precondition

$$G \wedge p_{G_n}$$

and effect p_G.

In either case, the formulation remains unchanged, all that is needed is to operate on the definition of the planning task at hand.

6.9 Summary

In the present chapter we have reviewed a small yet significant and relevant part of the existing literature of plan recognition, covering the major developments in the field between 1978 and today. We are conscious that this is a partial view of the field, and the reader may find missing some published work.

We note that we have consciously limited ourselves to review in depth only those works we found to offer something new to the field and which can be seen regularly cited in the most recent literature in plan recognition as a key influence. We also wanted to discuss at least an example for each of three major "families" of formulations. Namely, model–based and generative, as the one subject of this thesis, those relying on probabilistic inference over a plan library compiled into a graphical model, and library–based, generative approaches using parsing algorithms and grammars to solve and represent plan recognition problems.

[2]This is necessary in order to avoid bloating action definitions.

Chapter 7

Conclusions

We conclude offering a summary of the topics discussed in the previous chapters, as well as a summary of the contributions and a discussion of their significance. No research is ever finished and the one discussed in this thesis is no exception. Section 7.2 exposes the loose ends we see in our contribution and Section 7.3 discusses further immediate work that we find would round it up.

Chapters 2 and 3 have covered Classical, MDP and POMDP planning models, available modeling languages for describing real–world tasks in terms of such models, and algorithms for solving the models resulting from such descriptions. Chapter 4 has presented the formulation of plan recognition over classical planning domain theories, formally defining plan recognition problems and their solution, in terms of costs of classical plans. The domains used to evaluate the proposed approach have also been described with detail, closing with the results of the evaluation itself. Chapter 5 extends the formulation given in Chapter 4 to domain theories modeling partially–observable, stochastic environments. Formal definitions for plan recognition problems and solutions are adapted to such a setting, and algorithms given to find those solutions, that rely on MDP and POMDP solvers used in an off–the–shelf way. The reformulated framework is further illustrated with detailed descriptions of the benchmarks used to evaluate it. Finally, a selection of the most relevant previous work on plan recognition is reviewed in Chapter 6, which aims at highlighting both the influences and deep, significant differences between our proposed approach to the problem of plan recognition and others previously published.

7.1 Contributions

The main contribution of this thesis is a novel and crisp formulation of *single–agent*, *keyhole* (Kautz and Allen, 1986a) plan recognition as planning. Instances of plan recognition tasks are formalized in terms of probabilistic plan recognition theories $T = \langle P[\cdot], \mathcal{G}_T, O, Probs \rangle$ in Section 4.7. These theories account in a crisp and consistent way for a three elements common in all approaches to plan recognition.

The first element is the problem of representing knowledge about the observed agent possible actions, plans and goals. In our formulation this knowledge comes in the form of planning domains $P[\cdot]$, which are instances of formal planning models. These domains are encoded with planning languages which support *factored* representations of world states, action preconditions and effects. Past approaches to plan recognition (Schmidt et al., 1978; Perrault and Allen, 1980; Lesh and Etzioni, 1995; Baker et al., 2009) have introduced some or just one of the previously mentioned notions, but none includes all of them nor integrates planning models with varying expressive power in a coherent, comprehensive way using state–of–the–art planning languages.

The second element is the representation of observed agents *intentions*. In our formulation, these are encoded as a set of *hypothetical* goals $\mathcal{G}$ which are in turn logical formulas that denote *implicitly* sets of world states. This allows us to account in a simple, straightforward and elegant way with the problems posed by agents pursuing several goals simultaneously, by syntactic means alone. Accounting for constraints on how these goals are achieved is also possible to do by operating on $P[\cdot]$ modifying actions preconditions and effects, so that the set of possible trajectories over the state space is reduced to match those that satisfy such constraints. Also the definition of this hypothetical goal set $\mathcal{G}_T$ is not assumed to be restricted to some specific subsets of the possible state space, but can rather include *any* subset of states that satisfy the goal conditions.

The third element is that of the observed agent behavior. In our formulation observations are *totally ordered* sequences of actions O which are not assumed to be *complete* in the sense that between the actions observed there might be an unspecified number of actions done by the agent and not observed. Furthermore, no assumption is made about the *position* of the actions in O, these could be actions done at the beginning, end or the middle of agent performance.

A crisp definition of *solutions* to the plan recognition problems represented by theories T is given in terms of a *posterior goal probability* $P(G|O)$ distribution. This distribution is characterized within the framework of Bayesian inference appealing to the Bayes rule

$$P(G|O) = \alpha P(O|G)P(G) \tag{7.1}$$

where the prior goal probability distributions $P(G) \in \mathit{Probs}$ defined over goals $G \in \mathcal{G}_T$ are included as part of the theory to be solved. Observation sequence O likelihoods $P(O|G)$ are approximated by relying on the *principle of rationality* (Dennett, 1983) which in a planning context relies on the notion of agents being more likely to execute actions a so that the sum of the costs of the actions necessary to achieve a goal G is minimized. Agents with *bounded rationality* are considered by assuming them to execute actions according to a Boltzmann distribution with parameter β for controlling the degree in which agents deviate from optimal – minimum cost – performance.

Defining the precise details of $P(O|G)$ computation depends on the underlying planning model, and we give them for the case of classical planning models, with full–feedback from actions that have deterministic effects, as well as for Goal Mdp and Goal Pomdp models where feedback from actions is limited and action outcomes are non–deterministic. In any case, the particular computation is defined as a call to a planner, without requiring any change in the planner code.

Generative approaches to compute $P(G|O)$ are not either new, but previous formulations either use a much more limited representation of actions, states and goals, relying on algorithms whose assumptions are not entirely clear (Geib and Goldman, 2009), or they operate on explicit representations of the state space with much more stringent assumptions being made on the nature of observation sequences O (Baker et al., 2009).

Another contribution made is that of a set of ready to use, easily available benchmarks including the domains used to evaluate our formulation. A major problem when developing the research presented in this thesis has been the lack of publicly available benchmarks, as is the common practice in the automated planning community. We expect those benchmarks not only to illustrate our work, but also to provide other researchers with a solid foundation for evaluating their own work.

7.2 Loose Ends

While we consider our contribution to be addressing many of the issues and limitations hindering previous approaches to plan recognition, it is not free from limitations and loose ends. Leaving aside relatively trivial questions such as defining $P(O|G)$ for other planning models such as conformant and contingent planning (Cimatti and Roveri, 2000) or trying alternate planners on existing definitions of $P(O|G)$, we will focus on those we consider to be *fundamental*, and we do not know how to address them effectively at the time of writing this thesis.

Sensitivity of the Order of Magnitude Approximation

In Section 4.11 we have shown the approximation of $P(O|G)$ we use in Section 4.7 to be very robust and to have an accuracy proportional to the information conveyed by the observation sequence O. However, we could do potentially better by rather than just considering *one* of the best plans for $P[G+O]$ and $P[G+O]$, we rather considered the N best plans for each of these two planning problems.

This would allow to take into account the number of plans that actually satisfy O. Consider a planning domain where we have an agent navigating through a corridor represented by tiles arranged in 2 rows and m columns. The agent starts at the tile on the upper–right corner of the corridor, and she is expected to reach either the tile at positions $(2, n)$ or $(2, \frac{n}{2})$, corresponding to hypothetical goals G_1 and G_2 respectively. We observe the agent to move to the right from the starting tile.

When the priors for $P(G_1)$ and $P(G_2)$ are defined to be equal, both goals would have the same probability $P(G|O)$. However, if we recall that $P(O|G)$, when the agent is behaving *optimally*, is defined as

$$P(O|G) = \sum_{\pi} P(O|\pi)P(\pi|G)$$

then we can see that the sum over plans π will give a slightly higher value to goal G_1, since there are $m-1$ optimal plans satisfying O for goal G_1 and there are $\frac{m}{2}-1$ plans for goal G_2.

The problem lies in how to define, in a *principled* way, an approximation that would require $2|\mathcal{G}_T|$ calls to a classical planner in order to obtain a possibly more accurate approximation of $P(O|G)$. Yet another problem, is how to account for the probabilities $P(\pi|G)$ when agents are considered to be choosing actions according to a Boltzmann policy, that is, occasionally deviating from optimal behavior.

Partially–Ordered Observation Sequences

We could further relax our assumptions over the observation sequence O, so that rather than being a totally–ordered sequence of actions was instead a sequence of *partially–ordered* actions. The ability to handle such observation sequences would be relevant for plan recognition applications where the exact times at which actions were executed are only partially known.

This would require to come up with an alternate mapping to the one given in Definition 4.6. These partially ordered actions would induce a directed graph which would be needed to be mapped efficiently into actions conditional effects, so that valid plans for G should embed a serialization of the partially–ordered actions. We note as well that this graph does not need to be a tree, and could have cycles. Ambiguities also rise when actions appear mentioned more than once, and the partial orderings induce potentially many possible serializations.

7.3 Future Work

Other loose ends in the work described in this thesis, which can be readily overcome by either extending or reformulating part of the contributions presented, are next reviewed in the order we deem to go from the most concrete to the purely speculative.

Bridging the Gap Between Classical and MDP Accounts

The experimental evaluation in Section 5.7 shows the proposed formulation for plan recognition over Goal POMDP to be robust and to have an accuracy proportional to the amount of information conveyed by the observation sequence O. However, we are not entirely satisfied with the definition of $P(O|G)$ given in Section 5.5. The reasons for this are that we need an additional parameter m, the number of execution being sampled.

In the domains we have tested the approach based on sampling executions, we have found sampling variance to not be of much concern. But we cannot really guarantee that this will be as well the case for other domains. Also it is a concern that the definition for $P(O|G)$ in the classical setting takes into account the likelihood $P(\bar{O}|G)$ explicitly, while the definition for Goal MDPs and Goal POMDPs does in a *implicit* way.

An elegant way of doing away with sampling is that of estimating $P(O|G)$ and $P(\bar{O}|G)$ directly by mapping the problem of finding the maximum likelihood trajectory over states S_{π^*} for which the optimal value function $V^*_G(b)$ is defined, in to that of computing a minimum cost plan over a planning problem $P[G] = \langle F, I, A, c, G \rangle$ defined as follows:

1. A fluent set F where fluents represent states in S_π

$$F = \{p_s \mid s \in S_\pi\} \cup \{p_G\}$$

   where p_s are fluents that correspond to a state s which is visited by the greedy policy over V^*, and p_G is a dummy fluent that denotes that a goal state has been reached.

2. An action set A with actions accounting for state transitions that could occur when executing the Boltzmann policy

$$a_s : p_s \to p_{s'}$$

for each action a and states $s, s' \in S_\pi$ such that $P(s'|s,a) > 0$, along with an action to account for the fact that a goal state has been reached

$$a_{s,G} : p_s \to p_G$$

for states $s \in S_G$.

3. Action costs $c(a_s)$ accounting for the likelihood of selecting action a on state s and reaching state s'

$$-\log\{P(a|s,G) \cdot P(s'|s,a)\}$$

where $P(a|s,G)$ is the probability of selecting a on a state s when executing a Boltzmann policy.

4. With initial state $I = \{p_{s_0}\}$,
5. and goal state $G = \{p_G\}$.

Order of magnitude approximations of $P(O|G)$ and $P(\overline{O}|G)$ can be readily obtained from applying a slightly modified version of the mapping given in Definition 4.6 to compile observations into goals and conditional effects. We note the resulting planning problems $P[G+O]$, $P[G+\bar{O}]$ are delete–free and actions conform closely to the relaxation implicit in the h^1 heuristic. However it is not clear that classical planners will be able to process very efficiently with the potentially huge numbers of fluents and actions resulting from compiling MDPs and POMDPs with the mapping outlined above.

Agent POMDP Model Partially Known By the Observer

In our formulation, the observed agent model is known by the observer except for the hidden goal. Incomplete information about the *initial state* of the agent, however, can be accommodated as well. The simplest approach is to define a set $\mathcal{I}$ of possible initial states each with a probability $P(I)$. The formulation can then be generalized to deal with both hidden goals G and hidden initial belief states I, and the posterior probabilities over the collection of such pairs can be computed in a similar manner.

Failure To Observe Actions That Must Be Observed

As argued in (Geib and Goldman, 2009), information about actions a that if done, must always be observed. This information is valuable since the absence of such actions from O implies that they were not done. This information can be used in a direct manner in our formulation by adjusting the notion of when a plan π or execution τ complies with O. In the presence of *must-see* actions, plans π or executions τ comply with O when π (τ) embeds O, *and* every *must-see* action appears as many times in π (τ) as in O.

Observing What the Agent Observes

We have assumed that the observer gets a partial trace of the actions done by the agent and nothing else. Yet, in the setting of Goal POMDPs, if the observer gets to see some of the observation tokens $\omega \in Obs$ gathered by the agent, she can use this information as well. In particular, the number m_O in 5.7 would then be set to the number of sampled executions for G that comply with both O and Obs.

Noise In the Agent–Observer Channel

In the Goal POMDP setting, if the observer gets to see the actions done by the agent through a noisy channel where actions can be mixed up, then the problem of determining where a sample execution τ complies with the observations O is no longer a *boolean* problem where $P(O|\tau)$ is either 0 or 1. It rather becomes a probabilistic inference problem that can be solved in linear-time with Hidden Markov Model (HMM) algorithms, that would yield a probability $P(O|\tau)$ in the interval $[0, 1]$. For this, the model must be extended with probabilities $Q'(o|a)$ of observing token o from the execution of action a, and hidden chain variables $t_i = j$ expressing that the observation token o_i in $O = o_1, \ldots, o_n$ has been generated by action a_j in the sample execution $\tau = a_1, \ldots, a_m$.

Bibliography

Each reference indicates the pages where it appears.

C. Anderson, D. Smith, and D. Weld. Conditional effects in graphplan. In R. Simmons, M. Veloso, and S. Smith, editors, *Proceedings of the Fourth International Conference on AI Planning Systems (AIPS-98)*, pages 44–53. AAAI Press, 1998. 15, 45

D.E. Appelt and M. Luria. Planning natural language utterances. In *Proc. 1982 AAA/Conf*, volume 59, page 62, 1982. 109

K. Astrom. Optimal control of markov decision processes with incomplete state estimation. *J. Math. Anal. Appl.*, 10:174–205, 1965. 40

D. Avrahami-Zilberbrand and G. A. Kaminka. Fast and complete symbolic plan recognition. In *Proceedings of IJCAI*, pages 653–658, 2005. 3

C. Bäckström and B. Nebel. Complexity results for SAS$^+$ planning. *Computational Intelligence*, 11(4):625–655, 1995. 28

Christer Bäckström and Peter Jonsson. All pspace-complete planning problems are equal but some are more equal than others. In *SOCS*, 2011. 17

C. L. Baker, R. Saxe, and J. B. Tenenbaum. Action understanding as inverse planning. *Cognition*, 113(3):329–349, 2009. xvii, 85, 87, 88, 119, 120, 124, 125

J. Balcazar. The complexity of searching implicit graphs. *Artificial Intelligence*, 1 (86):171–188, 1996. 17

A. Barto, S. Bradtke, and S. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72:81–138, 1995. 36, 37, 41

R. Bellman. *Dynamic Programming.* Princeton University Press, 1957. 35

P. Bertoli, A. Cimatti, Marco Roveri, and Paolo Traverso. Planning in nondeterministic domains under partial observability via symbolic model checking. In *Proc. IJCAI-01*, 2001. 107

D. Bertsekas. *Dynamic Programming and Optimal Control, Vols 1 and 2.* Athena Scientific, 1995. 33, 34, 36

D. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, 1996. 37, 42

A. Blum and M. Furst. Fast planning through planning graph analysis. In *Proceedings of IJCAI-95*, pages 1636–1642. Morgan Kaufmann, 1995. 4, 24, 113

B. Bonet. An ϵ–optimal grid–based algorithm for partially observable markov decision processes. pages 51–58. Morgan Kaufmann, 2002. ISBN 1-55860-873-7. 42

B. Bonet and H. Geffner. Planning as heuristic search: New results. In *Proceedings of ECP-99*, pages 359–371. Springer, 1999. 20

B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 129 (1–2):5–33, 2001a. 4, 17, 21, 22

B. Bonet and H. Geffner. Gpt: A tool for planning with uncertainty and partial information. In *Proc. IJCAI Workshop on Planning with Uncertainty and Partial Information*, 2001b. 33, 43, 98

B. Bonet and H. Geffner. Labeled RTDP: Improving the convergence of real-time dynamic programming. In *Proc. 13th Int. Conf. on Automated Planning and Scheduling (ICAPS-2003)*, pages 12–31. AAAI Press, 2003. 33, 36, 38, 39, 40, 41, 43, 98

B. Bonet and H. Geffner. Solving POMDPs: RTDP-Bel vs. Point-based algorithms. In *Proceedings IJCAI-09*, pages 1641–1646, 2009. 35, 40, 41, 42

B. Bonet and M. Helmert. Strengthening landmark heuristics via hitting sets. In *Proceedings of ECAI 2010*, volume 215, pages 329–334, 2010. 20

B. Bonet, G. Loerincs, and H. Geffner. A robust and fast action selection mechanism for planning. In *Proceedings of AAAI-97*, pages 714–719. MIT Press, 1997. 17, 20

H. H. Bui. A general model for online probabilistic plan recognition. In *Proc. IJCAI-03*, pages 1309–1318, 2003. 3, 7, 93, 114, 116, 118

H. H. Bui, D. Phung, S. Venkatesh, and H. Phan. The hidden permutation model and location-based activity recognition. In *Proc. AAAI-08*, 2008. xxii, 73, 76, 78, 116

T. Bylander. The computational complexity of STRIPS planning. *Artificial Intelligence*, 69:165–204, 1994. 11, 16, 17, 22

E. Charniak and R. P. Goldman. A bayesian model of plan recognition. *Artificial Intelligence*, 64:53–79, 1993. 3, 8, 51, 113, 114, 116, 118

A. Cimatti and M. Roveri. Conformant planning via symbolic model checking. *Journal of Artificial Intelligence Research*, 13:305–338, 2000. 125

P. R. Cohen, C. R. Perrault, and J. F. Allen. Beyond question answering. In W. Lehnert and M. Ringle, editors, *Strategies for Natural Language Processing.* LEA, 1981. 3, 109

T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* The MIT Press, 1989. 16

K. Currie and A. Tate. O-Plan: the open planning architecture. *Artificial Intelligence*, 52(1):49–86, 1991. 113

D.C. Dennett. Intentional systems in cognitive ethology: The "panglossian paradigm defended". *Behavioral and Brain Sciences*, 6:343–390, 1983. 51, 87, 119, 124

A. Doucet, N. de Freitas, K. Murphy, and S. Russell. Rao-Blackwellised particle filtering for dynamic Bayesian networks. In *Proc. UAI-2000*, pages 176–183, 2000. 116

K. Erol, J. Hendler, and D. S. Nau. HTN planning: Complexity and expressivity. In *Proc. AAAI-94*, 1994. 73, 75, 117

T. Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, (27): 861–874, 2006. 99, 100

A. Fern, S. Natarajan, K. Judah, and P. Tadepalli. A decision-theoretic model of assistance. In *Proc. of AAAI*, volume 6, 2007. 96

G. Fey, J. Shi, and R. Drechsler. Efficiency of multi-valued encoding in sat-based atpg. In *Multiple-Valued Logic, 2006. ISMVL 2006. 36th International Symposium* *on*, pages 25–25. IEEE, 2006. 8

R. Fikes and N. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 1:27–120, 1971. 7, 13, 107, 109

L. R. Ford and D. R. Fulkerson. *Flows in Networks.* Princeton University Press, 1962. 22, 23

M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of AI Research*, 20, 2003. 15

B. Gazen and C. Knoblock. Combining the expressiveness of UCPOP with the efficiency of Graphplan. In *Proc. ECP-97*, pages 221–233. Springer, 1997. 15, 30, 31, 55

H. Geffner. Functional strips. In J. Minker, editor, *Logic-Based Artificial Intelligence*, pages 187–205. Kluwer, 2000. 7, 43

C. W. Geib and R. Goldman. Plan recognition in intrusion detection systems. In *Proc. DARPA Information Survivability Conference & Exposition (DISCEX-01)*, 2001. xxi, 73, 75, 76

C. W. Geib and R. P. Goldman. A probabilistic plan recognition algorithm based on plan tree grammars. *Artificial Intelligence*, 173(11):1101–1132, 2009. 3, 7, 8, 117, 118, 119, 124, 127

M. Goldszmidt and J. Pearl. Qualitative probabilities for default reasoning, belief revision, and causal modeling. *Artificial Intelligence*, 84(1-2):57–112, 1996. 63

E. Hansen and S. Zilberstein. Lao*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129:35–62, 2001. 36

P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.*, 4:100–107, 1968. 11, 20, 21

P. Haslum. Additive and Reversed Relaxed Reachability Heuristics Revisited, 2008. Proceedings of the 2008 Internation Planning Competition. 80

P. Haslum. hm (p)= h1 (pm): Alternative characterisations of the generalisation from hmax to hm. In *Proceedings ICAPS 2009*, pages 354–357, 2009. 24

P. Haslum and H. Geffner. Admissible heuristics for optimal planning. In *Proc. of the Fifth International Conference on AI Planning Systems (AIPS-2000)*, pages 70–82, 2000. 21, 24

P. Haslum, B. Bonet, and H. Geffner. New admissible heuristics for optimal planning. In *Proc. AAAI-05*, 2005. 21, 25, 80

P. Haslum, J. Slaney, and S. Thiébaux. Incremental lower bounds for additive cost planning problems. In *HDIP 2011 3rd Workshop on Heuristics for Domain-independent Planning*, page 15, 2011. 24

M. Hauskretch. Value–function approximations for partially observable markov decision processes. *Journal of Artificial Intelligence Research*, 13:33–94, 2000. 42

M. Helmert. Complexity results for standard benchmark domains in planning. *Artificial Intelligence*, 143(2):219–262, 2003. 17

M. Helmert. A planning heuristic based on causal graph analysis. In *Proc. ICAPS-04*, pages 161–170, 2004. 17

M. Helmert. New complexity results for classical planning benchmarks. In *Proceedings of the 6th International Conference on Automated Planning and Scheduling (ICAPS–06)*, pages 52–62, 2005. 11, 17

M. Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006. 4, 26, 28

M. Helmert and R. Mattmüller. Accuracy of admissible heuristic functions in selected planning domains. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, pages 938–943, 2008. 19

J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001. 4, 15, 17, 20, 22, 25, 26, 28

J. Hoffmann, J. Porteous, and L. Sebastia. Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, 2004. URL `http://www.jair.org`. 26, 27

J. Hoffmann, I. Weber, and F. M. Kraft. Sap speaks pddl. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, 2010. 7

H. H. Hoos. Sat-encodings, search space structure, and local search performance. In *International Joint Conference on Artificial Intelligence*, volume 16, pages 296–303. Citeseer, 1999. 8

M. J. Huber, E. H. Durfee, and M. P. Wellman. The automated mapping of plans for plan recognition. In *Proc. UAI-94*, pages 344–351, 1994. 3

L. P. Kaelbling, M. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1999. 40

H. Kautz. A formal theory of plan recognition and its implementation. In *Reasoning about Plans*, pages 69–124. Morgan Kauffmann Publishers, 1991. xxii, 3, 4, 110, 111, 112, 113, 114, 116

H. Kautz and J. F. Allen. Generalized plan recognition. In *Proc. AAAI-86*, pages 32–37, 1986a. 3, 8, 51, 123

H. Kautz and J. F. Allen. Generalized plan recognition. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 32–38, 1986b. 110

H. Kautz and B. Selman. Unifying SAT-based and Graph-based planning. In T. Dean, editor, *Proceedings IJCAI-99*, pages 318–327. Morgan Kaufmann, 1999. 4

H. A. Kautz and B. Selman. Planning as satisfiability. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI'92)*, pages 359–363, 1992. 113

E. Keyder and H. Geffner. Heuristics for planning with action costs revisited. In *Proceedings 18th European Conference on Artificial Intelligence (ECAI-08)*, 2008. 25

E. Keyder and H. Geffner. Trees of shortest paths vs. steiner trees: Understanding and improving delete relaxation heuristics. In *Proc. 21st Int. Joint Conference on AI (IJCAI-09)*, 2009. 24

E. Keyder, S. Richter, and M. Helmert. Sound and complete landmarks for and/or graphs. pages 335–340. 26

B. Knox and P. Stone. Combining manual feedback with subsequent mdp reward signals for reinforcement learning. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, volume 1, pages 5–12, 2010. 7

A. Koller and M. Stone. Sentence generation as a planning problem. In *Annual Meeting Association For Computational Linguistics*, volume 45, page 336, 2007. 109

A. Kolobov, D.S. Weld, et al. Retrase: integrating paradigms for approximate probabilistic planning. In *Twenty-First International Joint Conference on Artificial Intelligence*, 2009. 46

R. Korf. Depth-first iterative-deepening: an optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985. 20

R. Korf. Real-time heuristic search. *Artificial Intelligence*, 42:189–211, 1990. 40

R. Korf. Linear-space best-first search. *Artificial Intelligence*, 62:41–78, 1993. 19

M. Lekavý and P. Návrat. Expressivity of Strips-like and HTN-like planning. In *Proc. 1st KES Int. Symp.KES-AMSTA 2007*, pages 121–130, 2007. 7, 74

N. Lesh and O. Etzioni. A sound and fast goal recognizer. In *Proc. IJCAI-95*, pages 1704–1710, 1995. xvii, 3, 113, 115, 124

L. Liao, D.J. Patterson, D. Fox, and H. Kautz. Learning and inferring transportation routines. *Artificial Intelligence*, 171(5-6):311–331, 2007. 71

V. Lifschitz. On the semantics of STRIPS. In M. Georgeff and A. Lansky, editors, *Proc. Reasoning about Actions and Plans*, pages 1–9. Morgan Kaufmann, 1986. 13

J. McCarthy. Circumscription – a form of non–monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980. 112

J. McCarthy and P.J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine intelligence*, 4(463-502):288, 1969. 13

D. McDermott. Using regression-match graphs to control search in planning. *Artificial Intelligence*, 109(1-2):111–159, 1999. 21

D. McDermott. The 1998 AI Planning Systems Competition. *Artificial Intelligence Magazine*, 21(2):35–56, 2000. 13, 15, 29, 30

G. Monahan. A survey of partially observable markov decision processes: Theory, models and algorithms. *Management Science*, 28(1):1–16, 1983. 40

K. P. Murphy. Dynamic bayesian networks. *Probabilistic Graphical Models*, 2002. 116, 117

A. Newell and H. Simon. *Human Problem Solving.* Prentice–Hall, Englewood Cliffs, NJ, 1972. 11

N. Nilsson. *Principles of Artificial Intelligence.* Tioga, 1980. 11, 20

R. Nissim, J. Hoffmann, M. Helmert, et al. Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. In *HDIP 2011 3rd Workshop on Heuristics for Domain-independent Planning*, page 5, 2011. 20

J. Orkin. Agent architecture considerations for real-time planning in games. *Proceedings of the 1st AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2005. 7

C. M. Papadimitriou. *Computational complexity.* Addison-Wesley, Reading, Massachusetts, 1994. ISBN 0201530821. 16, 17

J. Pearl. *Heuristics.* Addison Wesley, 1983. 11, 17, 18, 19, 20, 21, 22

J. Pearl. *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, 1988. 113

E. Pednault. ADL: Exploring the middle ground between Strips and the situation calcules. In R. Brachman, H. Levesque, and R. Reiter, editors, *Proc. KR-89*, pages 324–332. Morgan Kaufmann, 1989. 54

W. Pentney, A. Popescu, S. Wang, H. Kautz, and M. Philipose. Sensor-based understanding of daily life via large-scale use of common sense. In *Proc. AAAI-06*, 2006. 3

C. R. Perrault and J. F. Allen. A plan-based analysis of indirect speech acts. *Computational Linguistics*, 6(3-4):167–182, 1980. 3, 13, 51, 53, 109, 110, 112, 124

G. Pezzulo. Coordinating with the future: The anticipatory nature of representation. *Minds & Machines*, (18):179–225, 2008. 119

J. Pineau, G. Gordon, and S. Thrun. Anytime point-based approximations for large pomdps. *JAIR*, 27:335–380, 2006. 42

M. Puterman. *Markov Decision Processes – Discrete Stochastic Dynamic Programming.* John Wiley and Sons, Inc., 1994. 34

D. V. Pynadath and M. P. Wellman. Probabilistic state-dependent grammars for plan recognition. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 507–514, 2000. 3, 116

M. Ramirez and H. Geffner. Plan recognition as planning. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-09)*, pages 1778–1783. AAAI Press, 2009. xvii, 49

M. Ramirez and H. Geffner. Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, 2010. xvii, 49

M. Ramirez and H. Geffner. Goal recognition over pomdps. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11)*, 2011. xvii

S. Richter and M. Westphal. The LAMA planner: using landmark counting in heuristic search, 2008. Proceedings of the 2008 International Planning Competition. 4, 17, 20, 21, 27, 68

S. Richter and M. Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39:127–177, 2010. 20, 28

S. Richter, M. Helmert, and M. Westphal. Landmarks revisited. In *Proc. AAAI*, pages 975–982, 2008. 26, 27, 80

V. Rieser and O. Lemon. Natural language generation as planning under uncertainty for spoken dialogue systems. *Empirical methods in natural language generation*, pages 105–120, 2011. 109

R. Rosen. *Anticipatory Systems.* Pergamon Press, 1985. 119

S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003. 16, 19

E. Sacerdoti. The nonlinear nature of plans. In *Proceedings of IJCAI-75*, pages 206–214, Tbilisi, Georgia, 1975. 108, 109

C. Schmidt, N. Sridharan, and J. Goodson. The plan recognition problem: an intersection of psychology and artificial intelligence. *Artificial Intelligence*, 11: 45–83, 1978. xv, xvi, xvii, xxiv, 3, 13, 51, 53, 107, 108, 109, 110, 119, 124

D. Silver and J. Veness. Monte-carlo planning in large pomdps. *Advances in Neural Information Processing Systems (NIPS)*, 2010. 46

E. Sondik. *The Optimal Control of Partially Observable Markov Processes.* PhD thesis, Stanford University, 1971. 40

E. Sondik. The optimal control of partially observable markov decision processes over the infinite horizon: discounted costs. *Operations Research*, 26(2), 1978. 40

R. Sutton and A. Barto. *Introduction to Reinforcement Learning.* MIT Press, 1998. 42

M. Vilain. Getting serious about parsing plans: A grammatical analysis of plan recognition. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 190–197, 1990. 117

T. Walsh. Sat v csp. *Principles and Practice of Constraint Programming–CP 2000*, pages 441–456, 2000. 8

D. S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4): 27–61, 1994. 110

J. Wu, A. Osuntogun, T. Choudhury, M. Philipose, and J.M. Rehg. A scalable approach to activity recognition based on object use. In *Proc. of ICCV-07*, 2007. xxii, 73, 77, 78, 79

Q. Yang. Activity Recognition: Linking low-level sensors to high-level intelligence. In *Proc. IJCAI-09*, pages 20–26, 2009. 3