Modeling and Solving Sequential Decision Problems with Uncertainty and Partial Information

Blai Tirant Bonet Bretto

January 2004

Technical Report R-315
Cognitive Systems Laboratory
Department of Computer Science
University of California
Los Angeles, CA 90095-1596, USA

This report reproduces a dissertation submitted to UCLA in partial satisfaction of the requirements for the degree of Doctor of Philosophy in Computer Science. This work was supported in part by AFOSR grant #F49620-01-1-0055, NSF grant #IIS-0097082, California State MICRO grant #00-077, and MURI grant #N00014-00-1-0617.

© Copyright by
Blai Tirant Bonet Bretto
2004

Contents

List of Figures

List of Tables

Abstract of the Dissertation

Modeling and Solving Sequential Decision Problems with Uncertainty and Partial Information

by

Blai Tirant Bonet Bretto
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 2004
Professor Judea Pearl, Co-chair
Professor Richard Korf, Co-chair

The idea of developing General Problem Solvers (GPS) has been in Artificial Intelligence (AI) since its inception. A GPS is a computer system comprised of a general description language and algorithms used to model and solve sequential decision problems. This dissertation focuses in developing a GPS aimed at sequential decision problems involving uncertainty and partial information.

The general approach proposed here addresses the task of building a GPS from three different and orthogonal dimensions of (i) mathematical models that characterize the problems, their solutions and optimal solutions, (ii) high-level description languages used to describe the mathematical models in an economical and amenable way, and (iii) general algorithms, based on heuristic search, used to solve the resulting models.

This work starts from the mathematical models of control theory and proposes novel representation techniques and algorithmic solutions superior to the ones traditionally used. The new algorithms are the result of casting the general sequential decision problem as a search problem in a suitable space that is then tackled using basic principles from AI. The dissertation thus presents a new perspective on sequential decision making that allows for a fruitful cross-fertilization between control theory and AI.

The contributions of this work are diverse. We show new theoretical results about the existence and characterization of solutions for the mathematical theory of Markov decision processes.

On the algorithmic side, we propose new algorithms based on heuristic search that compute partial optimal policies, together with new heuristic functions. We show the correctness and complexity of the algorithms, and the formal properties of the heuristic functions. The performance of the new algorithms is shown to be superior, through a number of experiments, to the current state-of-the-art algorithms.

On the representation side, we introduce a new and very expressive representation language for planning problems involving uncertainty and partial information. This is one of the first general representation languages capable of coping with a broad range of planning problems including robot navigation, game playing, synthesis of circuits, medical diagnosis, sequential knowledge acquisition, etc.

Several extensions of the approach are also included. They give formal characterization of models known up to a certain degree of precision, and novel proposals for trading off sub-optimality of solutions against performance.

These extensions are built upon methods from qualitative decision theory (as the kappa functions proposed by Spohn and used by Pearl and others to model qualitative information), methods of propositional satisfiability, and the novel data structures used in formal verification.

Acknowledgments

I would like to give warm and sincere thanks to my doctoral committee of extraordinary professors. First, to Judea Pearl that welcomed me into his research group where I was given complete freedom and support to develop my research. Prof. Pearl is an incredible scientist that always surprises me with his wisdom, insight, intuition and suggestions. Few minutes with him were always enough to clear the mind, and see the forest whenever I was lost wandering between the trees. Second, to Richard Korf, another incredible scientist, for sharing his knowledge and expertise, and the many suggestion he gave me. His office's door was always open, and he welcomed me into his office every time (even when I just dropped without notification). Prof. Korf's research, talks and classes were a bottomless source of ideas and motivation for my research. I thank both Judea Pearl and Richard Korf, the co-chairs of my committee, for their constant confidence and support even in times when they didn't know my whereabouts.

Third, thanks to Adnan Darwiche for creating the AI seminar at UCLA where I was given multiple opportunities to present my research. Such presentations allowed me to improve my work and presentations skills. Prof. Darwiche is an incredible active researcher that irradiates knowledge and enthusiasm to all AI students at UCLA, not only his students.

Fourth, thanks to Thomas Liggett that taught me all I know about formal mathematical analysis and probability theory. I was very fortunate to have Prof. Liggett as the professor of the first math class I took at UCLA. Prof. Liggett is an extraordinary professor that gives simple and clear explanations, never makes a mistake, challenges and motivates the students, and always has time, inside and outside the classroom, to answer all questions. My first experience with Prof. Liggett was so enjoyable that I ended up taking five graduate courses in probability theory at the UCLA math department.

Fifth, thanks to Héctor Geffner. Héctor is my former master's advisor and now a best friend and my closest collaborator. He was my mentor that introduced me to scientific research, show me how to do research, and is the main responsible for my decision to pursue doctoral studies. Through the years, he has been a infinite source of wonderful ideas, motivation, guidance and support. Héctor is perhaps the wisest person that I know.

I would like to thank other people at the UCLA computer science department. Thanks to Prof. Stott Parker and Prof. Milos Ercegovac for their support and confidence. Thanks to Kaoru Mulvihill for the help, friendship, and candies. Thanks to Verra Morgan for solving all the messes I made, keeping me out of trouble, and her friendship. And thanks to Roberta Nelson and Peter Schultze for all their help.

My studies, research, and trips were supported by a USB/CONICIT fellowship from Venezuela, and grants from NSF, ONR, AFOSR, and the DoD MURI program.

Through the years, I have the opportunity to meet colleagues with similar research interests and to have illuminating discussions with them. Thanks to Sylvie Thiébaux, Jörg Hoffmann, Alessandro Cimatti, Jussi Rintanen, Enrico Giunchiglia, Fahiem Bacchus, Ronen Brafman, Rina Dechter, Valentina Bayer, Michael Littman, Craig Boutilier, Derek Long, Maria Fox, and others.

Many thanks to my friends at the office: Carlos Brito, Jin Tian, Mark Hopkins and Chen Avin for their friendship, uncountable many discussions at every lunch, and making my life easier in Los Angeles. Thanks to friends at UCLA and Los Angeles: Shailesh Vaya, Akash Nanavati, Douglas Carroll, Murali Mani, Keith Cascio, Fang (Andrea) Chu, Thomas Phan, Jean El-Khoury, and Rafael Carbunaru and his future wife Nataly. Thanks to my friends from Venezuela and their constant support: José Ramón Sabino, Harold Sotillo, Miguel Ángel Yánez, Pedro Córdova, and Ricardo Salamé and all their wives.

Thank very much to my girlfriend Natalia Caamaño for all her love and support, for all the patience and the many round trips Caracas-Los Angeles. Undoubtedly, I would have not finished without all your help.

Finally, thanks to my family, my parents Roger Bonet and Yolanda Bretto, my brothers Marc and Grau and his wife Eva, my nephew Marcel, and my grandmother Yaya, for all the unconditional support and love.

Chapter 1

Introduction

The idea of developing *General Problem Solvers* (GPS) has been in Artificial Intelligence (AI) since its inception [151]. A GPS is a computer system comprised of a general description language for specifying sequential decision tasks and methods that are used to solve such tasks.

Initially, the main motivation behind developing a GPS was to understand the complexities of the human thought *by imitating human thinking* [152]. Nowadays, the aspirations are more modest and the main motivation is to develop general solvers capable of *achieving performance similar* to that of hand-crafted solvers for specific problems. More precisely, the main goal is to close the performance gap to the point where the tradeoff between ease of specification and solving time is no longer an important issue.

An achievement of this sort has already occurred in the field of computer languages. For example, it is no secret that any implementation in assembly language has better performance than an implementation in a high-level programming language like $C++$ or *Java*. Yet, the effort needed for the former task is so large compared to the gain in performance, that nobody develops in assembly language any more. All this is possible because of the tremendous advances in computer architecture, compilers and code optimization techniques.

It was clear since the beginning of AI that to develop a GPS its *scope* must be clearly defined. Otherwise, the task becomes so general and ill-defined that nothing can be said or done. The scope of a GPS can be bounded by means of the mathematical models implemented by the GPS. The model pertaining to classical planning, for example, is the deterministic state model in which the initial state is fully known and actions map states to states deterministically. Thus, the solution to a classical planning problem is just a sequence of steps that achieves the goal from the initial state. As the planning task becomes more complex, however, the deterministic state model is no longer appropriate and more complex models become necessary.

Consider for example the popular game of Mastermind, produced by Invicta Plastics, Ltd., that enjoyed a wave of popularity during the Christmas of 1975 [1]. As we will see, the game of Mastermind is a good motivating example that illustrates some of the more complex domains needed when modeling and solving sequential decision tasks.

Mastermind is a two player game in which the first player, designated as the codemaker, chooses a secret code comprised of a sequence of 4 colored pegs from a set of six different colors that has to be discovered by the second player or codebreaker. The second player moves consist of guesses presented to the first player which are then 'marked' according to how close they match the secret code.

Precisely, the first player marks each guess with 4 smaller black and white pegs. A black marker means that one of the pegs in the guess is in the right position and of the right color, while a white marker means that one peg in the guess is of the right color but in the wrong position. The positions of the markers are not relevant but it is customary to present all black markers before the white ones.

In the original version of the game, the codebreaker is given 10 chances to discover the code, and the game is finished either when the codebreaker runs out of chances, in which case the codemaker wins, or when the the last guess is marked with 4 black pegs, in which case the codebreaker wins.

This game has not only captivated thousands of players around the world, but it has attracted the interest of some mathematicians [118, 110, 50, 83], and is closely related to the 12-coins problem originally posed by Erdös and Renyi

secret code		markers
$(\square, \square, \square)$	$\to$	$\{\bullet, \bullet\}$
$(\square, \square, \times)$	$\to$	$\{\bullet, \bullet, \bullet\}$
$(\square, \times, \square)$	$\to$	$\{\bullet, \circ, \circ\}$
$(\square, \times, \times)$	$\to$	$\{\bullet, \bullet, \circ\}$
$(\times, \square, \square)$	$\to$	$\{\bullet, \circ, \circ\}$
$(\times, \square, \times)$	$\to$	$\{\bullet, \bullet, \circ\}$
$(\times, \times, \square)$	$\to$	$\{\circ, \circ, \circ\}$
$(\times, \times, \times)$	$\to$	$\{\bullet\}$

Table 1.1: Possible answers after a first guess of $(\square\,\square\,\times)$ in a Mastermind game with three pegs and two colors.

[75]. (In the 12-coins problems, the challenge is to discover a counterfeit coin of different weight from a set of 12 identically-looking coins by performing weighings in a balance scale. It is known that the counterfeit coin can be detected with at most 3 weighings and at the same time deciding if the coin weights less or more than a regular coin.)

For simplicity, consider an instance of Mastermind consisting of just 3 pegs and 2 colors that are denoted with the symbols $\times$ and $\square$. After a first guess of $(\square, \square, \times)$ there are only 6 possible marks that the first player can generate which are given in Table 1.1. For example, if the secret code is $(\times, \times, \square)$ and the first guess is $(\square, \square, \times)$, then the markers are $\{\circ, \circ, \circ\}$ since the three symbols in the guess appear in the code but in different positions.

Given this description of Mastermind, our goal is to compute an *optimal universal strategy* which unveils the secret code in a minimum number of guesses no matter what the value of the secret code. In other words, the goal is to compute an strategy for Mastermind that minimizes the worst possible case. For the above example, Fig. 1.1 shows an strategy for the game with a worst-case cost of 2 that is clearly optimal since the code cannot be always discovered with just one guess.

With respect to planning tasks, the Mastermind game falls into the category of non-deterministic contingent sequential decision tasks and corresponds to an instance of the mathematical model known as *Partially Observable Markov Decision Processes* (POMDPs).

The most substantial difference between these models and the deterministic ones is in the form of the solution. Indeed, whereas for the latter models a solution is a sequence of steps, for the former it is a policy or strategy that needs to consider every possible answer from the first player to the posed guesses.

A second example of the kind of problems considered in this dissertation is the Omelette problem due originally to [132]. The problem consists of an agent that has a large supply of eggs and whose goal is to get 3 good eggs and no bad ones into one of two bowls. The eggs can be either good or rotten, and the agent can perform the actions of grabbing an egg, breaking an egg into a bowl, pouring the content of a bowl into another, and cleaning a bowl. The agent can also sense if a bowl contains a rotten egg or not by smelling at it. The uncertainty in this domain comes from the fact that each egg can be good or rotten with probability $1/2$ respectively and that such status is not known to the agent in advance.

A difference with the the game of Mastermind is that in this task the objective is to compute an optimal strategy that achieves the goal with minimum *expected* number of steps instead of minimum worst-case number of steps. This subtle difference however is an important one since it can be shown that minimizing worst-case scenario is not well defined in this problem. Indeed, Fig. 1.2 shows an optimal policy with achieves the goal with an expect cost of 23 steps, yet observe that the number of steps is not uniformly bounded across all possible trajectories, i.e. that for each integer n there exist trajectories with positive probability that take more than n steps. Hence, a worst-case analysis in this problem would assign infinite cost to all strategies so that no discrimination among strategies would be possible.

Part of this dissertation is dedicated to characterizing which problems have well-defined 'optimal' solutions with respect to worst-case and/or expected cost scenarios.

The policies Fig. 1.1 and Fig. 1.2 were obtained with a general algorithm based on heuristic search that uses an *automatically-deduced* heuristic function from the description of the problem.

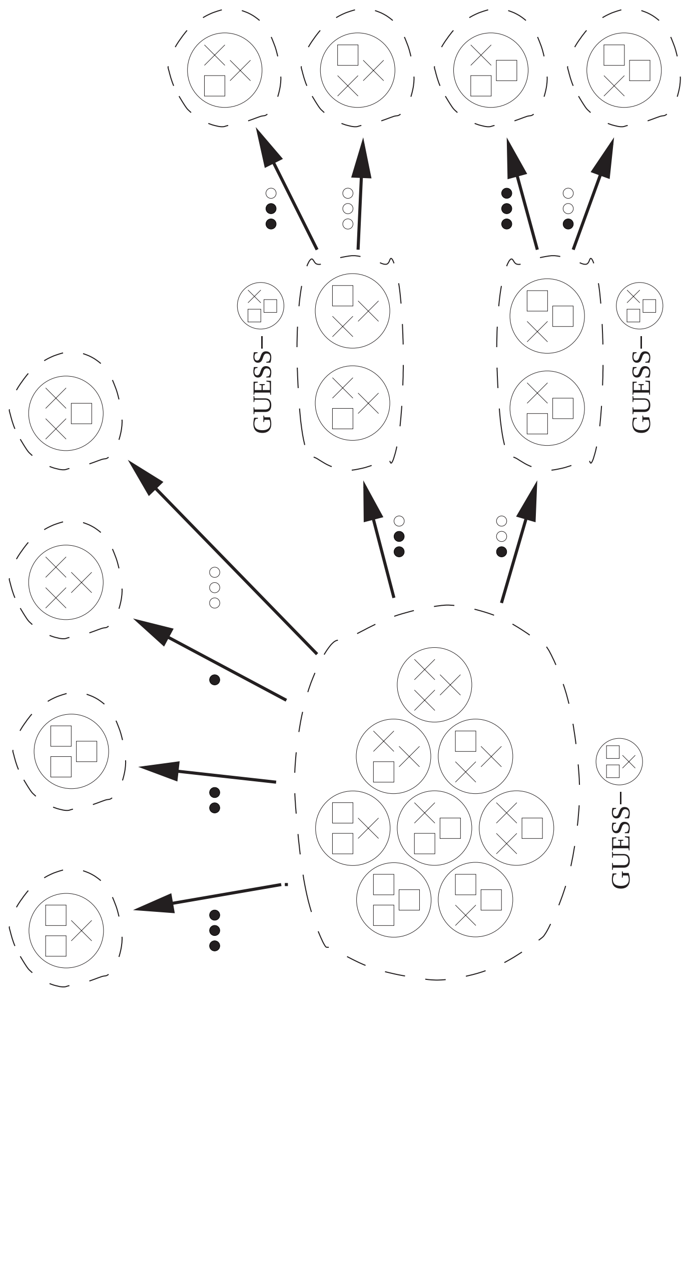


Figure 1.1: Optimal strategy for Mastermind with 3 pegs and 2 colors.

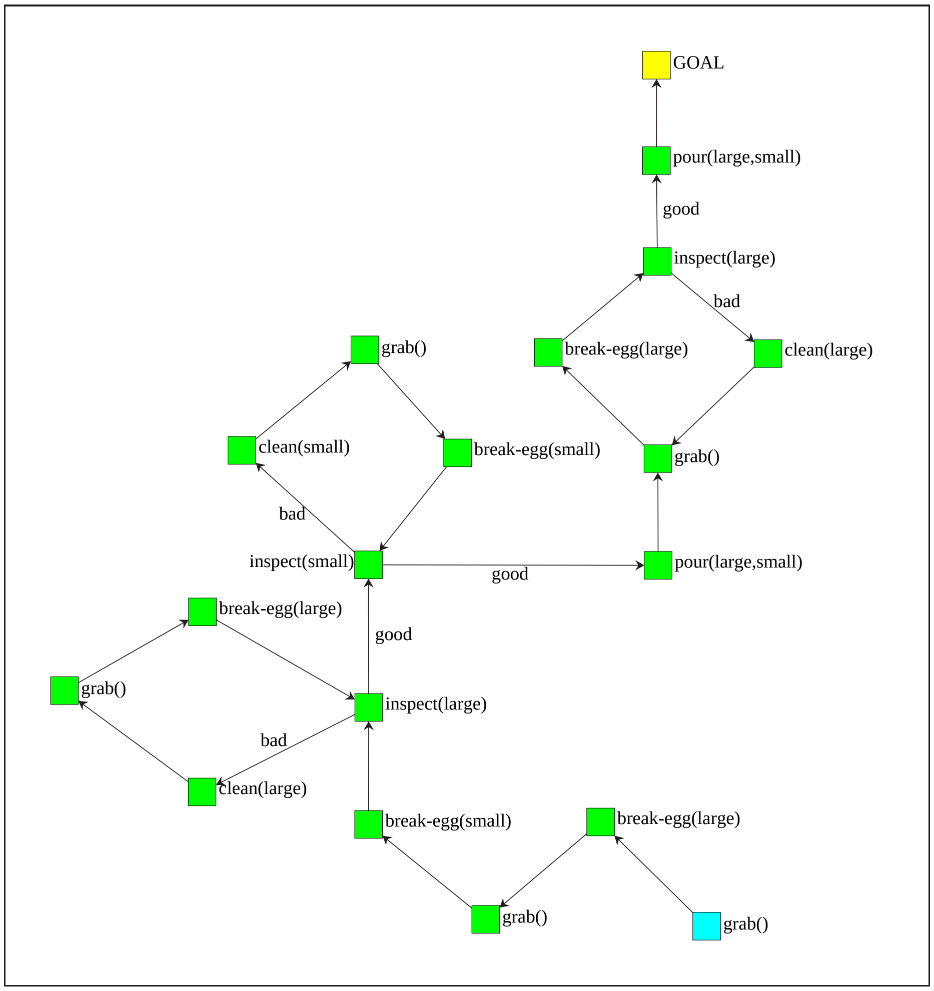


Figure 1.2: Optimal policy for the omelette problem with three eggs.

It is well known that standard search problems can be very hard to solve due to the combinatorial explosion of the possibilities. Yet, efficient algorithms and powerful heuristics have been developed for solving difficult combinatorial problems. It is our belief that most of the lessons learned in standard heuristic search can be applied successfully in this setting in order to design algorithms that are more efficient and powerful than the current state-of-the-art algorithms for this class of problems.

Thus, this dissertation develops a unified and coherent framework for modeling and solving sequential decisions problems involving uncertainty and partial information using techniques borrowed from heuristic search.

More specifically, the current work focuses on a *domain-independent* approach to the problem of finding optimal strategies for problems specified in a high-level description language. The term domain independent refers to the fact that all the information that can be used to solve a given problem must come (or be inferred) from the description of it.

Rather than only focusing on building a specific GPS, this dissertation also provides the general foundations and principles needed for such development; all of them that are partitioned into three categories:

- **Mathematical Models** for defining the different search spaces, the solutions to these spaces, and the characterization of optimal solutions,
- **High-Level Languages** for describing these models economically and in a way amenable to people, and

- **Algorithms** based on heuristic search for finding solutions to these models.

This dissertation elaborates each of these topics separately and shows how from general and basic principles one can build a GPS with crisp semantic notions of what can be modeled, what can be solved, and the reasons why some problems cannot be modeled or solved.

Among the contributions of this work are new theoretical results about the different mathematical models, new representation languages and new general algorithms and heuristics that improve the current state of the art.

Most of these ideas are then implemented into a single planning system which is tested over some difficult benchmark problems. The experimental results show the scope and limitations of the approach together with directions of current and future research.

One short note about the scope of this work is that neither models with continuous time, durative actions or continuous state spaces are ever considered. The closest to these things that this dissertation ever gets is in the case of general partial information models which are transformed to a special case of a continuous state space model.

This dissertation is organized as follows.

In Ch. 2, the general planning problem and a brief characterization of the different models that have been studied in AI is given together with their relation to the mathematical models. Thus, this chapter sets up the general context in which all other chapters are developed.

Ch. 3 offers a thorough exposition and analysis of the different mathematical models relevant to the planning task. The goal of this chapter is to lay down the formal framework of analysis and to make this dissertation self contained. Some novel results and proofs of already known theorems are also included in this chapter.

The description language for general planning tasks involving uncertainty and partial information is given in Ch. 4. The language is explained in a friendly way through a set of examples and then its semantics is given in terms of the mathematical models. The dependency between this chapter and the previous one is only necessary for understanding the formal semantics of the language; a strong requirement if an implementation of a GPS based on such a language is desired. The reader that is only looking for a syntactical description of the language together with an intuitive knowledge of the semantics can skip Ch. 3 and go directly to Ch. 4.

Ch. 5–7 present known and new algorithms for solving the different mathematical models. The chapters go over the most important standard algorithms, some recent algorithms relevant to our work, and the new algorithms developed in the context of the current work. The chapters present formal proofs of the correctness of the new algorithms as well as their complexities. Additionally, some experimental results over benchmark problems show the practical benefits of the proposed algorithms.

Most of the new algorithms are based on heuristic search and so they need heuristics functions to perform well. Different domain-independent heuristic functions for each type of model are given in Ch. 8 together with their most important formal properties.

Most of the ideas presented in chapters 1–8 are implemented in a single GPS system called GPT for General Planning Tool. The GPT system is then evaluated into a number of different planning problems in Ch. 9.

One issue that emerged during the research associated with this dissertation was the fact that sometimes one is interested in solving a problem that is only known up to a certain degree of specification. This task falls into the field of qualitative decision making which is a mainstream area in AI.

There are different well-established formalisms for representing and combining qualitative quantities like probabilities and utilities. Among them, ordinal ranking functions, also known as kappa rankings, have shown to be very effective and well understood. In addition, some of the initial research on kappa rankings was done by Judea Pearl and students in the Cognitive Systems Laboratory group here at UCLA, so I have adopted their approach in order to model sequential decision tasks that are only known up to certain degrees of specification.

Ch. 10 presents an extension of the mathematical models to the case of systems specified using kappa rankings. The chapter presents formal definitions for such systems, characterization of solutions and optimal solutions and some algorithmic techniques. The resulting theory, that we call an Order-of-Magnitude approximation to sequential decision tasks, is given in this chapter.

Ch. 11 presents other extensions to the basic theory whose main objective is to tradeoff solution quality versus performance. Thus, in this chapter we relax the requirement of obtaining optimal policies and present some preliminary research that makes this tradeoff in a principled way. Two different approaches are given in this chapter together with some experimental results that show both proposals are very promising.

In the first one, a general scheme that interleaves planning and execution is presented in which the agent computes an initial partial plan that only considers the most plausible scenarios, and only replans when the system gets out of such ‘envelope’. This approach combines ideas found in the algorithms together with kappa rankings for quantifying different degrees of plausibility.

In the second approach the idea is to reduce the search space into a smaller space made of equivalence classes, i.e. the search space is factored using an equivalence relation. Once this reduction is made, any algorithm can be used to find a solution to the factored space. Then, this solution has to be transformed into a suitable solution for the original space. For this approach, we show how the reduction can be done and computed in a principled way, and give a characterization of when a solution to the factored space is a solution to the original space.

This dissertation finishes with two chapters. The first contains open directions for future research in the different areas of mathematical models, languages and algorithms, and the second contains the overall conclusions and contributions of this work. Finally, three appendices with the mathematical background, a formal syntax of the planning language and the description of the benchmark problems are also included.

* * *

As for reading purposes, an effort has been made to make the chapters as independent as possible from the others, and as a result different readings can be performed.

A brief reading focused on the mathematical aspects of the approach can be obtained by going over chapters 2, 3, 12, and further extended with chapters 10 and 11.

A second reading focused on the algorithms can be obtained with chapters 2, 5–7, 11 and 12.

Those readers more interested in planning concepts, the description language, and empirical results, can go with chapters 2, 4, 9 and 12, and can also include 11 for an overview of recent representation techniques.

Finally, for those brave enough, there is the ‘full tour’ that is nothing else but a cover-to-cover trip.

Chapter 2

Planning Models

This chapter presents the different planning tasks developed in AI through the years and their relation to the mathematical models. As it will be seen, these models result from combining different transition functions, sensor models, and observability assumptions.

The problem of selecting actions in environments that are dynamic and not completely predictable or observable is a central problem in intelligent behavior. In AI, this translates into the problem of designing controllers that can map sequences of observations into actions so that certain goals can be achieved.

Three main approaches have been used in AI for designing controllers to such planning tasks:

- the *programming* approach, where the controller is programmed by hand in a suitable high-level procedural language,
- the *planning* approach, where the controller is derived automatically from a description of the actions, sensors, goals, etc., and
- the *learning* approach, where the controller is derived from data based on previous experiences.

The work of R. Brooks and others in mobile robotics is an example of the first approach [41, 3]. Control theory and AI planning are examples of the second approach [154, 82], and systems that learn by trial and error or generalization are instances of the third approach [190, 20].

The three approaches exhibit both strengths and limitations. In the programming approach, for example, multiple procedures can be programmed for achieving different goals at different levels of granularity, and then combined into an efficient hierarchy of subtasks that can decompose a complex problem into simpler subproblems. The limitation, of course, is that a badly chosen decomposition leads to inefficient and often unfeasible solutions. Hence, choosing the right decomposition is crucial, a task whose solution is rarely obvious.

This work focuses on the *planning approach*. More specifically, it presents a general and operational approach to planning that combines elements from AI planning, dynamic programming and logic, and is able to deal with systems that exhibit different types of *dynamics* (deterministic, non-deterministic, or probabilistic) and different types of *feedback* (null, partial, or complete). Plans, or solutions, are thus *open-loop* or *closed-loop* controllers, that make decisions based on previous actions and observations, according to the type of sensors available; see Fig. 2.1.

The main ingredients of the approach are the use of *high-level logical representation languages* for describing the actions, sensors, and goals, *mathematical models of sequential decisions* for precisely describing the various planning tasks, and *heuristic search algorithms* for obtaining their solutions.

For the transition dynamics, the following possibilities are considered:

- *Deterministic* transitions, characterized by a deterministic transition function $f(s,a)$ that maps a state s and an applicable action a into the resulting state,
- *Non-deterministic* transitions, characterized by a non-deterministic transition function $F(s,a)$ that maps a state s and an applicable action a into a set of possible resulting states, and

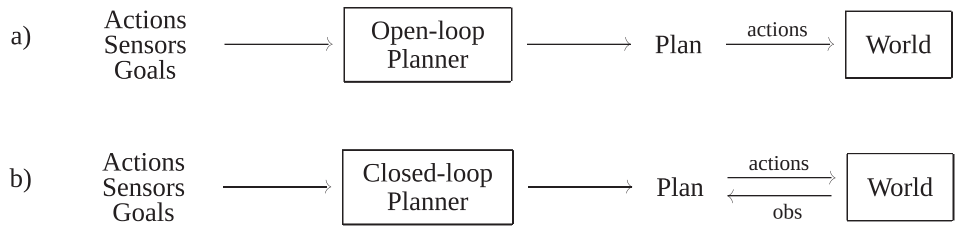


Figure 2.1: Open and closed-loop planning: in open-loop planning (a) the plans is a fixed sequence of actions, while in closed-loop planning (b), the sequence depends on the observations and previous actions.

- *Stochastic* transitions, characterized by a probability transition function $p(s, a, s')$ that is the probability of the next state being s' after applying action a in state s.

In the case of deterministic transitions, if the initial state is known at the beginning, then all future states are completely determined by the sequence of actions, so it is sufficient to only consider open-loop controllers. On the other hand, if the initial state is not known or there is uncertainty about the outcome of the actions, the future states are not predictable and closed-loop controllers need to be considered.

Although non-deterministic transitions might be treated as a special case of stochastic transitions, the current work discriminates them by considering different cost criteria. While in the non-deterministic case the interest is in minimizing the worst-case scenario, in the stochastic case the interest is in minimizing the expected behavior. This distinctions will become clear later when formalizing the different models.

The feedback or sensing information is what the *controller* receives from the environment after applying actions. From an formal point of view, they are modeled as an abstract set O of possible observations or signals. The current agreement in AI and control theory is to deal with sensor models given by a function $O(s, a) \subseteq O$ such that $O(s, a)$ stands for the set of possible observations that can be obtained when the system enters the state s after the application of the action a; that is, the formalization only considers models in which the observations depend on the current state of the system and the last applied action. Other more general sensor models, e.g. that observations can depend on the state where the action was applied, have been considered, yet such models can be reduced to the present case by extending the description of the states.

Thus, the present work focuses on the following possibilities for the sensor model:

- *Complete* sensing, that corresponds to perfect information and is characterized by the function $O(s, a) = \{s\}$,
- *Null* sensing, that corresponds to a blind controller and is characterized by the condition $O(s, a) = O(s', a')$ for all states s, s' and respectively applicable actions a, a',
- *Partial non-deterministic* sensing, in which the observations do not identify the state *uniquely* but after a feedback $o \in O(s, a)$ the controller knows that the system is *not* in states s' such that $o \notin O(s', a)$, and
- *Partial stochastic* sensing, in which the possible observations are characterized by a stochastic sensor model $q(s, a, o)$ which is the probability of receiving observation o after entering the state a as a result of applying action a.

An example of the third case is given in the game of Mastermind in which the state of the system is given by the secret code and the actions only return information. Following the example of the introduction, if the secret code (state) is $s = (\Box, \times, \Box)$ and the guess (action) is $a = (\Box, \Box, \times)$ then $O(s, a) = \{\{\bullet, \circ, \circ\}\}$. Thus, if feedback $\{\bullet, \circ, \circ\}$ is obtained after the guess $(\Box, \Box, \times)$, the codebreaker knows that the secret code is either $(\Box, \times, \Box)$ or $(\times, \Box, \Box)$.

Similar to the case of deterministic transition functions with known initial state, in the case of null feedback there is no information available from which to make decisions; that is, it is sufficient to only focus on open-loop controllers. In this case, a sequence of actions must attain the goal no matter what is the evolution of the system.

		Transition Function		
		Deterministic	Non-deterministic	Probabilistic
Sensor	Complete	Classical	Non-deterministic	Probabilistic
	Partial	Contingent	Contingent	Contingent
	Null	Conformant	Conformant	Conformant

Table 2.1: The different planning models along the two dimensions of transition functions and sensor models.

planning model	classical	non-deterministic	probabilistic
mathematical model	deterministic in state space	non-deterministic MDP	stochastic MDP

planning model	conformant	non-deterministic contingent	probabilistic contingent
mathematical model	deterministic in belief space	non-deterministic POMDP	stochastic POMDP

Table 2.2: Relation between the planning and mathematical models.

Transition dynamics and sensor models constitute orthogonal dimensions of features that when combined generate different models of planning. The only conditions imposed on the possible combinations is that *non-deterministic transition models must not be combined with stochastic sensor models, and vice versa*. This restriction, that arises since a non-deterministic component cannot be combined plainly with an stochastic one, should cause no further trouble.

Now, we can characterize the different planning models that have been studied in AI. The case of classical planning, that has been studied extensively in AI, corresponds to a model with complete knowledge of the initial state and a deterministic transition function [152, 82, 140, 200]. Probabilistic and non-deterministic planning models, which correspond to a complete sensor model and a non-deterministic or stochastic transition function, have been studied in AI [129, 60, 139, 190], reinforcement learning [190, 114], and control theory under the name of Markov Decision Processes (MDPs) [164, 16, 18]. Models with null feedback and non-deterministic or stochastic transitions have appeared recently under the name of *conformant* planning problems or non-observable MDPs [185, 51, 13]. Finally, the most general case of a partial sensor model and arbitrary transitions correspond to non-deterministic and stochastic contingent planning tasks. Such models have appeared recently in AI [162, 77, 163, 69, 113, 170], control theory and operations research with the name of Partially Observable MDPs (POMDPs) [5, 187, 137]. This relationship between the transition function, sensor models and classes of planning problems is summarized in Table 2.1.

All these formalizations are instances of the general *state model* found in [152, 153]. In the case of complete feedback, for example, the appropriate states are simply the set of system states, whereas in the case of null or partial feedback, the states to consider are *belief states* (a term that refers to the collection of states that the controller deems possible at a certain time).

The mathematical models used to characterize the planning models and their solutions are the theory of Markov Decision Processes and Partially Observable Markov Decision Processes. These theories are originally from operations research where they have been used to model and solve general optimization problems. AI has recently turned its attention to such models since they provide a crisp framework for representing a large number of important problems as, for example, robot navigation and localization, medical diagnosis, stochastic games, market trading and bidding, automatic synthesis of circuits and programs, etc. The relation between the planning and mathematical models is shown in Table 2.2. Their study is the purpose of the next chapter while the relation between high-level description languages and the mathematical models is given in Ch. 4.

2.1 Summary

This chapter presents the most important classes of planning models that have been considered in AI. The models are classical (deterministic) planning, conformant planning, and contingent planning, all of which correspond to special cases of a general model defined along the two dimensions of transition dynamics and sensor models. These relations are shown in Table 2.1.

Other planning approaches are those that combine domain-independent planning techniques with hand-crafted or 'learned' knowledge that is used to control the search; e.g. [148, 76, 68, 7].

Chapter 3

Mathematical Models

This chapter presents the mathematical models used to characterize the different planning tasks. The models are Deterministic Decision Processes, Markov Decision Processes in their non-deterministic and stochastic versions, and Partially Observable Markov Decision Processes also in their non-deterministic and stochastic versions.

Also, important results for these models like existence of solutions, characterization of optimal solutions and some convergence properties are established. Some of the results are new proofs of already known theorems and others are completely new. For example, we give formal definitions and results for non-deterministic problems, something that does not appear in the standard references for the mathematical models; e.g. [164, 16]. New proofs of the existence, characterization and convergence for stochastic system is given. For the case of stochastic shortest-path problems, we provide new results for system with infinite state spaces under reasonable assumptions like the existence of a proper policies. In the case of POMDPs, we show some topological properties of the optimal value function that are used in Ch. 7 to develop the first optimal algorithm for POMDPs based on discretizations of the belief space.

Some parts of this chapter require some knowledge of mathematical analysis. Most of the needed concepts are summarized in Appendix A.

The different mathematical models that characterize the planning tasks are generalizations of the simple deterministic *state models* whose basic ingredients are:

S1. A *finite* state space S,

S2. a collection $A(s) \subseteq A$ of applicable actions for each state $s \in S$, and

S3. action costs $g(s, a)$ for each state $s \in S$ and action $a \in A(s)$.

State models were used since the beginning of AI to model a variety of tasks including constraint propagation systems, game playing and planning [82, 152, 153]. Although such models are also common in other disciplines like control theory and operations research, a subtle difference with respect to AI is that the actions (or controls) are qualified as applicable or not given a state. This distinction, that seems to be original from AI [82], allows for greater flexibility when making abstractions, a task that is fundamental to AI. A more detailed discussion of this issue with an example is given in Ch. 4.

The formalizations will be built upon these ingredients plus others in order to define classes of *controllable transition systems* or transition systems for short. In its more abstract form, a transition system is simply a triplet (S, A, Ω) where S is a state space, A is a function $s \rightsquigarrow A(s)$ that maps states to sets of applicable actions, and Ω is a collection of possible *trajectories* or *executions*. That is to say Ω is a subset of S^∞ where S^∞ is the infinite set product $S \times S \times S \times \cdots$. As we will see, not every sequence of states is a valid trajectory since the applicable actions A and the transition function impose constraints upon such trajectories. Constraints that are specified below for each specific model.

Similarly, the cost function g is used to assign costs to each valid trajectory of the system specifically for each model. Thus, the relation between trajectories, applicable actions and costs is given later for each specific model.

Some general definitions about trajectories are the following. A trajectory $(s_0, s_1, \dots) \in \Omega$ is called an s-trajectory if $s_0 = s$. The state s is called the initial state of the trajectory and the set of all s-trajectories is denoted by

Ω_s. Clearly, $\Omega = \cup_{s \in S} \Omega_s$ and $\Omega_s = \Omega \cap (\{s\} \times S^\infty)$.

Intuitively, a trajectory $(s_0, s_1, \dots)$ is an abstract representation of the evolution of a system in *discrete* time in which s_k represents the state of the system at time $t = k$, and s_0 is the initial state at time zero. When speaking about a given system, $X = (X_0, X_1, \dots)$ denotes a variable that ranges over the possible evolutions of the system in time, and each X_k is a state variable that represents the state of the system at time k. Finally, the chunk $(X_0, \dots, X_k)$ denotes a partial trajectory from time 0 to time k.

Given a specific transition system, the main goal is to define the set of *solutions* to the system and the set of *optimal* solutions to the system. The form of the solutions and the optimality criteria varies with each model as shown below. For example, sometimes we are interested in minimizing the worst-case cost scenario whereas other times the interest is to minimize the average-case cost scenario.

3.1 Deterministic Decision Processes

The simplest type of transition systems are the ones associated with deterministic actions. They are characterized by

D1. A finite state space S,

D2. finite set of actions $A(s) \subseteq A$ for each state $s \in S$,

D3. a *known* initial state $s_{init} \in S$,

D4. a set of *goal* states $S_G \subseteq S$,

D5. a *deterministic* transition function $f(s, a)$ for $s \in S$ and $a \in A(s)$ that is equal to the state that results of applying action a in state s, and

D6. *positive* action costs $g(s, a)$ associated with each $s \in S$ and $a \in A(s)$.

In the case of the puzzle known as Rubik's cube, for example, the state space consists of all possible configurations of the cube, the set of actions the 18 possible actions of rotating a face 90, 180 or 270 degrees, the initial state a known initial configuration of the cube, the set of goal states is the single configuration in which each face has unique color, and the action costs are equal to 1. Clearly, for each action (face rotation), there is only one possible result of applying such action in a given state. In this example, all the sets $A(s)$ are equal to the collection of the 18 possible rotations. As we will see, since the costs are all equal to 1, the task here is to compute a minimum-length sequence of actions that map the initial configuration to the goal configuration.

In the rest of this section, the ingredients D1–D6 are used to define a specific transition system (S, A, Ω), the set of solution to such system and the set of its optimal solutions.

The transition system is given by the set S, the function A, and the collection Ω of trajectories given by

$$(s_0, s_1, \dots) \in \Omega \quad \text{iff} \quad s_0 = s_{init}, \forall(k \geq 0)\exists(a \in A(s_k))(s_{k+1} = f(s_k, a)) .$$

That is, a valid trajectory is one in which each state s_{k+1} is the result of applying an action $A(s_k)$ in the state s_k. Transition systems of this form are called Deterministic Decision Processes or DDPs for short.

Given that the initial state is known and the determinism of the system, a solution to a model like D1–D6 is a sequence of actions $(a_0, a_1, \dots, a_n)$ that generates a (partial) trajectory $(s_0 = s_{init}, s_1 = f(s_0, a_0), \dots, s_{n+1} = f(s_n, a_n))$ such that each action a_k is applicable in s_k, and s_{n+1} is a goal state; i.e. $a_k \in A(s_k)$ and $s_{n+1} \in S_G$. The solution is defined to be optimal when the total cost $\sum_{k=0}^{n} g(s_k, a_k)$ is minimum. In classical planning, it is often assumed that all costs $g(s, a)$ are equal to one so that an optimal solution is one of minimum *length*.

An important characterization of the optimal solutions can be given using Bellman's Principle of Optimality which says:

Any sub-path of an optimal path must be optimal.

This property, which is a consequence of the additive nature of the costs, translates into equation form as follows. Consider a function $J : S \to \overline{\mathbb{R}}$, which shall be interpreted as that $J(s)$ is the cost of achieving a goal state from state s. We say that J is *consistent* at state s if the value $J(s)$ is consistent (with respect to the interpretation) with the values of all neighbors of s, and that J is consistent if it is consistent at every state. In symbols, $J : S \to \overline{\mathbb{R}}$ is consistent if and only if it satisfies the system of equations:

$$J(s) \;=\; \begin{cases} 0 & \text{if } s \in S_G, \\ \min_{a \in A(s)} \; g(s,a) + J(f(s,a)) & \text{if } s \notin S_G, \end{cases} \tag{3.1}$$

for all states $s \in S$. If J^* denotes the minimum consistent function (i.e. for any consistent function J, $J^*(s) \leq J(s)$ for all $s \in S$), then a sequence of actions $(a_0, a_1, \ldots, a_n)$ is optimal if and only if its associated trajectory $(s_0, s_1, \ldots, s_{n+1})$ satisfies $s_{n+1} \in S_G$ and

$$J^*(s_k) \;=\; \sum_{l=k}^{n} g(s_l, a_l), \quad k = 0, \ldots, n+1.$$

Thus, there is a correspondence between the function J^* and the set of optimal solutions. In fact, given the value of J^*, we can recover an optimal solution by choosing actions greedily with respect to J^* as follows:

$$\begin{aligned} a_0 &\overset{\text{def}}{=} \operatorname*{argmin}_{a \in A(s_0)} \; g(s_0, a) + J^*(f(s_0, a)), \\ a_1 &\overset{\text{def}}{=} \operatorname*{argmin}_{a \in A(f(s_0,a_0))} \; g(f(s_0,a_0), a) + J^*(f(s_0,a_0), a)), \\ &\vdots \end{aligned}$$

Such sequence is called a *greedy* sequence with respect to J^*. (The proof of the existence and uniqueness of J^* as well as the relation between optimal solutions and greedy sequences is given later in the more general context of non-deterministic systems.)

This characterization of optimal solutions is the basis of the well-known algorithms of Dijkstra and Bellman-Ford for computing shortest paths in graphs [53].

The DDP *problem* is defined as the problem of finding an optimal sequence of actions. The equations (3.1) are known as Bellman's optimality equations and J^* is called the optimal value function with respect to such equations. As shown, finding J^* is sufficient for recovering an optimal sequence of actions.

3.2 Markov Decision Processes

In the presence of non-deterministic actions, i.e. actions that might have more than one outcome, the evolution of the system is uncertain and the model needs to be extended. This section defines two possible extensions by considering non-deterministic and stochastic transition systems under the following assumption:

A0. States are fully observable.

3.2.1 Non-Deterministic Systems

The simplest treatment for non-deterministic actions is to consider the possible outcomes as just possibilities without introducing any notion of chance or probability. This case corresponds to a transition system generated by:

N1. A finite state space S,

N2. finite set of actions $A(s) \subseteq A$ for each state $s \in S$,

N3. the known initial state $s_{init} \in S$,

N4. a set of *goal* states $S_G \subseteq S$,

N5. a *non-deterministic* transition function $F(s, a)$ for each $s \in S$ and $a \in A(s)$ that is equal to the set of possible successor states after applying a in s, and

N6. action costs $g(s,a)$ associated with each $s \in S$ and $a \in A(s)$.

A model like N1–N6 defines a transition system with Ω given by

$$(s_0, s_1, \ldots) \in \Omega \quad \text{iff} \quad s_0 = s_{init},\, \forall(k \geq 0)\exists(a \in A(s_k))(s_{k+1} \in F(s_k, a))\,.$$

Thus, if $X = (X_0, X_1, \ldots, X_t)$ denotes a *partial* execution up to time t, then the possible next states only depend on X_t and not in the whole X. This property, which corresponds to a memory-less system, is known as the *Markov* property, and the process is referred as a *Markov Decision Process* or MDP. The Markov property plays a fundamental role in the analysis of these systems.

Observe that given a state and an applicable action, the possible next states are not *predictable* and thus, a solution cannot be a sequence of actions but a function that maps states into actions. A *strategy* or *policy* π is an *infinite* sequence $(\mu_0, \mu_1, \ldots)$ of *decision functions* where μ_k maps states into actions so that the action $\mu_k(s)$ is applied in state $X_k = s$ at the time $t = k$, the only restriction being $\mu_k(s) \in A(s)$ for all $s \in S$. If $\pi = (\mu, \mu, \ldots)$ the policy is called *stationary*, i.e. the control does not depend on time, and it is simply denoted by μ. If it is further assumed that goal states are *absorbing*, that is $g(s,a) = 0$ and $F(s,a) = \{s\}$ for all $s \in S_G$ and $a \in A(s)$, then the *cost function* associated with policy π when the system starts at state s is defined as[1]

$$J_\pi(s) \overset{\text{def}}{=} \max\Big\{ \limsup_{N\to\infty} \textstyle\sum_{k=0}^{N} g(s_k, \mu_k(s_k)) : (s_0 = s, s_1, \ldots) \in \Omega_\pi \Big\}$$

where Ω_π is the set of π-trajectories, i.e.

$$(s_0, s_1, \ldots) \in \Omega_\pi \quad \text{iff} \quad \forall(k \geq 0)(s_{k+1} \in F(s_k, \pi(s_k)))\,.$$

Having defined the cost of a policy, an optimal policy is one which achieves minimum cost; that is, an optimal policy π^* is one such that for any other policy π, the inequality $J_{\pi^*}(s) \leq J_\pi(s)$ holds for all $s \in S$. In the case that an optimal policy exists, J_{π^*} is denoted by J^* and called the optimal value function.

Optimal policies (and in general policies that always achieve a goal state) are not guaranteed to exist unless some assumptions are imposed. The following are perhaps the weakest assumptions that guarantee the existence of solutions:

A1. There exists a policy π and a (uniform) bound N_π such that for all trajectory $(s_0, s_1, \ldots) \in \Omega_\pi$, the state $s_k \in S_G$ for some $k \leq> N_\pi$.

A2. All costs $g(s,a)$ are positive except for the goal states, whose costs are all zero.

Any policy that satisfies the condition in A1 is called a *proper* policy. Given these assumptions is not hard to show that a policy is improper if and only if it incurs in an infinite cost for at least one state.

The (non-deterministic) MDP *problem* is defined as to find an optimal policy for a model like N1–N6 that satisfies these assumptions. The following result establishes the existence of solutions and provides a characterization of them in terms of optimality equations. The proof is given at the end of this section. This result seems to be known for some researchers but to my knowledge it does not appear explicitly in the literature. A similar result for stochastic MDPs is given in the next section.

Theorem 1 *Assume a transition system given by N1–N6 under assumptions A0–A2. Then, (i) there exists an optimal policy π^*, (ii) there exists an optimal policy μ^* that is stationary (which is a greedy policy with respect to J^*), and (iii) the optimal value function J^* is the unique (finite valued) solution to the Bellman's equations:*

$$J(s) = \min_{a \in A(s)} \left(g(s,a) + \max_{s' \in F(s,a)} J(s') \right). \tag{3.2}$$

[1]See the entry [A4] in Appendix A for the definition of $\limsup$. In this case, if all costs are non-negative the $\limsup$ equals $\lim$. From now on, a cross-reference like [A4] refers to an entry in Appendix A on mathematical background.

Remarks. Note that a stationary policy μ is greedy with respect to a value function J if and only if $T_\mu J = TJ$. If the transition function $F(s,a)$ is a singleton for all $s \in S$ and $a \in A(s)$, then the transition system is a simple deterministic system. Thus, this theorem establishes the claims made earlier about DDPs.

As the theorem asserts, the greedy policy μ^* with respect to J^* defined as

$$\mu^*(s) \overset{\text{def}}{=} \operatorname*{argmin}_{a \in A(s)} \left(g(s,a) + \max_{s' \in F(s,a)} J^*(s') \right)$$

is an optimal policy. Thus, finding an optimal policy is equivalent to solving Bellman's equations.

The optimal value function, and its greedy policy, *solves* the non-deterministic MDP problem from any possible *initial state*, s_{init} inclusive. Later, in Ch. 6, methods that focus only on solutions *closed* with respect to a fixed initial state are considered.

Proofs

The proof of the theorem will be given with the help of two operators that map $\mathbb{R}^{|S|}$ into itself that are called the Bellman operators. The importance of such kind of operators for the general theory of dynamic programming was first noted in [65]. In this non-deterministic setting and for a stationary policy μ, the operators are:

$$(T_\mu J)(s) \overset{\text{def}}{=} g(s,\mu(s)) + \max_{s' \in F(s,\mu(s))} J(s'),$$

$$(TJ)(s) \overset{\text{def}}{=} \min_{a \in A(s)} \left(g(s,a) + \max_{s' \in F(s,a)} J(s') \right).$$

Lemma 2 (Monotonicity) *Let J, J' be two $|S|$-dimensional real vectors such that $J \le J'$; i.e. $J(s) \le J'(s)$ for all $s \in S$. Then, $T_\mu J \le T_\mu J'$, $TJ \le TJ'$ and $TJ \le T_\mu J$.*

Proof:

$$(T_\mu J)(s) = g(s,\mu(s)) + \max_{s' \in F(s,\mu(s))} J(s') \le g(s,\mu(s)) + \max_{s' \in F(s,\mu(s))} J'(s'),$$

and the expression on the right is $(T_\mu J')(s)$. Similar arguments show the other two inequalities. □

Lemma 3 *Assume a model like N1–N6. Then, for $J_0 \overset{\text{def}}{=} 0$ and any policy $\pi = (\mu_0, \mu_1, \dots)$, $J_\pi = \limsup_{k\to\infty} T_{\mu_0} T_{\mu_1} \cdots T_{\mu_k} J_0$.*

Proof: The principle of induction shows the validity of

$$\max_{X \in \Omega_\pi \cap \Omega_s} \sum_{k=0}^{N} g(X_k, \mu_k(X_k)) = (T_{\mu_0} \cdots T_{\mu_N} J_0)(s)$$

for general state s and integer N. The result follows by taking the $\limsup$ in both sides. □

Proof Theorem 1: If $\pi = (\mu_0, \mu_1, \dots)$ is the proper policy, then Lemmas 2 and 3 imply that for M a bound on the costs, $0 \le (T^{k+1} J_0)(s) \le (T_{\mu_0} \cdots T_{\mu_k} J_0)(s) \le J_\pi(s) \le N_\pi M$ where T^k is the k-fold composition of T with itself and J_0 is the value function identically zero. Thus, by Lemma 2, $\{T^k J_0\}_{k\ge 0}$ is a bounded and increasing sequence, so it converges to a non-negative and finite limit J^* with $J^*(s) \le N_\pi M$ [A3].

By the continuity of T, $TJ^* = T \lim_{k\to\infty} T^k J_0 = \lim_{k\to\infty} T^{k+1} J_0 = J^*$ which establishes that J^* is a solution to (3.2). If μ^* is a greedy policy with respect to J^*, then $T_{\mu^*} J_0 \le T_{\mu^*} J^* = TJ^* = J^*$ which implies (with a similar argument) that μ^* is a proper policy with some constant N_{μ^*}. The uniqueness of the solution comes from the equality $T^k J = J^*$ for all $k \geq N_{\mu^*}$ and all vectors J such that $J(s) = 0$ for $s \in S_G$. A fact that follows from

$$(T^{k+1}J)(s_0) \;=\; \min_{a_0 \in A(a_0)} \cdots \max_{s_{k+1} \in F(s_k, a_k)} \sum_{l=0}^{k} g(s_l, a_l) + J(s_{k+1}) \;=\; J^*(s_0)$$

since, by assumption A1, after N_{μ^*} steps the state s_k must be a goal state. Therefore, if J is another solution to Bellman's equation, $J = T^{N_{\mu^*}+1} J = J^*$. To finish, we need to show that μ^* is an optimal policy. Fix a policy $\pi = (\mu_0, \mu_1, \ldots)$. If it is shown that

$$T^k J_0 \;\leq\; T_{\mu_0} T_{\mu_1} \cdots T_{\mu_k} J_0, \quad k \geq 0,$$

then $J_{\mu^*} \leq J_\pi$ and the optimality of μ^* follows since π is arbitrary. But this inequality is a direct consequence of Lemma 2. $\square$

3.2.2 Stochastic Systems

In this case, the non-deterministic transition function $F(s, a)$ is replaced by transition probabilities $p(s, a, s')$ over states; i.e. $p(s, a, s')$ are non-negative numbers such that $\sum_{s' \in S} p(s, a, s') = 1$. Thus, a stochastic transition system is characterized by

S1. A finite state space S,

S2. finite set of actions $A(s) \subseteq A$ for each state $s \in S$,

S3. the known initial state $s_{init} \in S$,

S4. a set of *absorbing* goal states $S_G \subseteq S$,

S5. transition probabilities $p(s, a, s')$ for $s \in S$ and $a \in A(s)$ which is equal to the probability that the next state is s' after applying action a in s,

S6. action costs $g(s, a)$ associated with each $s \in S$ and $a \in A(s)$, and

S7. a *discount factor* $\alpha \in [0, 1]$.

For an example of this type of system, consider the task of hanging a frame from the wall which main goal is to place a nail inside the wall by hitting it with a hammer. Assume that each hit of the nail with the hammer either succeeds and puts the nail in the wall or misses and leaves the nail intact, each possibility with probability $1/2$. This problem can then be modeled with two states $s =$ `nail-out` and $s' =$ `nail-in`, with one action $a =$ `hit-nail`, and with the transition probabilities $p(s, a, s) = p(s, a, s') = 1/2$ and $p(s', a, s') = 1$. The only goal state is just s' and all action costs are equal to 1.

In this setting, rather than to constraint the set of trajectories, Ω is defined as $S \times S \times \cdots$ and a family of probability measures is built over it.[2] As in the non-deterministic case, the objects of interests are policies $\pi = (\mu_0, \mu_1, \ldots)$ and their value functions J_π. The evolution of the system under policy π from an *initial* state s is governed by the probability measure $P_{\pi,s}$ defined as follows. Let $(s_0, s_1, \ldots, s_n) \in \Omega$ be a partial trajectory and $B \subseteq \Omega$ the *cylinder* set generated by it [A22]; i.e.

$$B \;\overset{\text{def}}{=}\; \{(s'_0, s'_1, \ldots) \in \Omega : s'_k = s_k, 0 \leq k \leq n\}.$$

Then, there exists a *unique* probability measure $P_{\pi,s}$ satisfying [A20]:

$$P_{\pi,s}(B) \;=\; \begin{cases} \prod_{k=1}^{n} p(s_{k-1}, \mu_{k-1}(s_{k-1}), s_k) & \text{if } s_0 = s, \\ 0 & \text{if } s_0 \neq s \end{cases}$$

[2]Throughout this dissertation, all the σ-fields associated with probability spaces [A20] are considered to be the ones generated by the cylinder sets [A22], and all random variables and functions can be shown to be measurable [A23]. Therefore, these nice technicalities would not be mentioned in future, yet the mathematically-inclined reader should be aware of them.

for all such cylinder sets. The measure $P_{\pi,s}$ characterizes the stochastic evolution of the system under the policy π when it starts at state s, and the cost function associated to π is defined as

$$J_\pi(s) \overset{\text{def}}{=} \limsup_{N\to\infty} E_{\pi,s}\left\{\sum_{k=0}^{N} \alpha^k g(X_k,\mu_k(X_k))\right\} \tag{3.3}$$

where the X_k's are random variables that stand for the state of the system at time k, and $E_{\pi,s}$ is the expectation with respect to $P_{\pi,s}$.

As can be seen, the discount factor is used to diminish future incurred costs at a geometric rate. Discount factors have a natural interpretation within the context of an inflationary economy but within the standard planning context they are only used to guarantee the existence of optimal strategies as is shown below.

The stochastic MDP *problem* is to find an *optimal policy* π^* satisfying $J_{\pi^*}(s) \le J_\pi(s)$ for every other policy π and state $s \in S$. If such π^* exists, its cost vector is called the optimal value function and is denoted by J^*. As before, the formal theory is based on the Bellman's operators given by:

$$(T_\mu J)(s) \overset{\text{def}}{=} g(s,\mu(s)) + \alpha \sum_{s'\in S} p(s,\mu(s),s')J(s'), \tag{3.4}$$

$$(TJ)(s) \overset{\text{def}}{=} \min_{a\in A(s)}\left(g(s,a) + \alpha \sum_{s'\in S} p(s,a,s')J(s')\right). \tag{3.5}$$

The following theorem, analogous to the one for the non-deterministic case, is the fundamental result about stochastic MDPs. A proof of this result can be found in [16], yet I have included the proof at the end of this section for purposes of completeness.

Theorem 4 *Assume A0 and a stochastic transition system defined by S1–S7. If $\alpha < 1$, then (i) there exists an optimal policy π^*, (ii) there exists an optimal policy μ^* that is stationary (which is the greedy policy with respect to J^*), and (iii) the optimal value function J^* is the unique (finite valued) solution to the Bellman's equations $TJ = J$.*

A greedy policy μ with respect to a cost function J is defined in this case as

$$\mu(s) \overset{\text{def}}{=} \operatorname*{argmin}_{a\in A(s)}\left(g(s,a) + \alpha \sum_{s'\in S} p(s,a,s')J(s')\right),$$

or equivalently μ is greedy with respect to J if and only if $T_\mu J = TJ$.

The discounted case $\alpha < 1$ is important since it allows for simple analysis of otherwise complex questions. Indeed, in this case the operator T becomes a contraction with coefficient α [A14], i.e. $\|TJ - TJ'\| \le \alpha\|J - J'\|$, and many properties can be easily obtained from this fact.

For example, using the contraction fact k times with J^* and any other vector J, it follows that

$$\|J^* - T^kJ\| = \|T^kJ^* - T^kJ\| \le \alpha^k\|J^* - J\|$$

and the reader can convince himself that T^kJ converges *geometrically* to J^* as $k \to \infty$.

Yet, perhaps the most important result is that a bound on the *suboptimality* of a greedy policy can be derived from the difference $\|TJ - J\|$. The suboptimality of a (stationary or not) policy π is defined as the quantity

$$\max_{s\in S} |J_\pi(s) - J^*(s)| = \max_{s\in S} J_\pi(s) - J^*(s).$$

Using the contraction property, the following result holds even in the case of infinite state spaces. Bertsekas [16] contains the proof for the case of finite state spaces that also works in this case.

Theorem 5 (Bertsekas [16]) *Consider a discounted MDP in an arbitrary state space S and an arbitrary value function J. Let μ be the greedy policy with respect to J, i.e. $T_\mu J = TJ$. Then, the following bound on the suboptimality of μ holds:*

$$\sup_{s\in S}\big[J_\mu(s) - J^*(s)\big] \le \frac{\alpha}{1-\alpha}\left(\sup_{s\in S}\big[(TJ)(s) - J(s)\big] - \inf_{s\in S}\big[(TJ)(s) - J(s)\big]\right).$$

Proofs

Proof of Theorem 4: In the case of $\alpha < 1$ the operator T becomes a contraction [A14] with respect to the max norm [A12] since

$$\begin{aligned}
|(TJ)(s) - (TJ')(s)| &= \Bigg| \min_{a \in A(s)} \Big(g(s,a) + \alpha \sum_{s' \in S} p(s,a,s')J(s') \Big) - \\
&\qquad \min_{a' \in A(s)} \Big(g(s,a') + \alpha \sum_{s' \in S} p(s,a',s')J'(s') \Big) \Bigg| \\
&\leq \alpha \Bigg| \sum_{s' \in S} p(s,a^*,s')J(s') - \sum_{s' \in S} p(s,a^*,s')J'(s') \Bigg| \\
&\leq \alpha \sum_{s' \in S} p(s,a^*,s')|J(s') - J'(s')| \\
&\leq \alpha \|J - J'\|
\end{aligned}$$

where a^* is the action that minimizes the smallest term in the equality. Therefore, T has a unique fix point J^*, and for any vector J, $T^k J \to J^*$ as $k \to \infty$. The rest of the proof is similar to the non-deterministic case. □

3.2.3 Stochastic Shortest-Path Problems

As in the non-deterministic case, some extra conditions are needed to guarantee the existence of optimal policies in the case of $\alpha = 1$. The following assumption, that is the stochastic version of that used for non-deterministic systems, is given in a slightly more general framework where the finiteness of the state space is not required.

A1. There exists a stationary policy μ and a constant N_μ such that

$$\rho_\mu \overset{\text{def}}{=} \sup_{s \in S} P_{\mu,s}(X_{N_\mu} \notin S_G) < 1. \tag{3.6}$$

Such policies are known as *proper policies* and the ones that are not proper are known as *improper policies*.

In words, a proper policy μ is a policy for which there exist a time horizon N and $\epsilon > 0$ such that the system enters a goal state with probability ϵ in at most N steps no matter what is the initial state.

In the case of a finite state space, the constant N_μ can be taken equal to $|S|$. Observe that since the goal states are absorbing, (3.6) also holds for any other integer greater than N_μ.

The other assumption is also similar to the non-deterministic case.

A2. All costs are positive and uniformly bounded away from zero except for the goal states, whose costs are all zero.

Systems satisfying assumptions A1–A2 are known as *Stochastic Shortest-Path* problems or SSPs, and are the subject of this section.

It is not hard to show that both assumptions imply that any improper policy generates a cost function that is not uniformly bounded; that is, $\sup_{s \in S} J_\mu(s) = \infty$ if μ is not proper.

Thus, both assumptions preclude cases where the optimal solution might 'wander' around without ever getting to the goal; for example, a problem with a zero-cost cycle in state space. This is the natural generalization of the corresponding assumptions for the non-deterministic case. In both cases, none of the assumptions can be completely removed.

Under the above assumptions, we are able to show the following theorem for stochastic shortest-path problems with infinite state spaces. This is a new theorem whose proof uses different techniques than the ones used for the case of finite state spaces. The version for finite state space appears (with a weaker assumption than A2) in [16], and a generalization of it for the case of infinite action sets appears in [17]. The proof of this theorem and other results appear at the end of this section.

Theorem 6 *Assume an undiscounted model S1–S7 and A0–A2 where the state space is not necessarily finite. Then, (i) there exists an optimal policy μ^* that is stationary, (ii) J^* is the unique solution to Bellman's equations, and (iii) for any vector J (in a compact set [A7]) with $J(s) = 0$ for goal states $s \in S_G$,*

$$\lim_{k\to\infty} T^k_{\mu^*} J \;=\; J_{\mu^*} \;=\; J^* \;=\; \lim_{k\to\infty} T^k J. \tag{3.7}$$

Remarks. In the case of infinite state spaces, the compactness requirement cannot be completely removed. The convergence in the first limit of (3.7) is *uniform* over the state space [A13] whereas for the second limit it is not. Of course, in finite state spaces, all convergences are uniform.

In the undiscounted case, the rate of convergence of $T^k J$ cannot be easily obtained as in the discounted case. The following theorem gives a geometric rate of convergence for a class of initial vectors in a *neighborhood* of J^* for the case of a *finite* state space. This is a new result whose proof is based on a coupling of two stochastic processes.

Theorem 7 *Assume an undiscounted model S1–S7 and A0–A2, and that the transition probabilities satisfy $p(s,a,s) = 0$ for all $s \in S \setminus S_G$ and $a \in A(s)$. If the integer $K > 0$ and real $0 \le \alpha < 1$ satisfy*

$$\frac{\|J^* - J\|_1 + \|J^*\|}{\underline{g}(K+1)} \;+\; \frac{\|J^* - J\|}{\underline{g}} \;\le\; \alpha, \tag{3.8}$$

where $\underline{g}$ is the minimum positive cost and $\|\cdot\|_1$ is the L_1 norm [A12]. Then, $\|J^* - T^{K+1}J\| \le \alpha\|J^* - J\|$ for all J such that $0 \le J \le TJ$.

Remarks. The extra assumption can be safely assumed since it is known that any SSP problem can be converted into an equivalent SSP problem satisfying it [16]. The theorem asserts that T^{K+1} is a *pseudo-contraction* with coefficient α over the set $\{J : 0 \le J \le TJ\}$ that satisfy (3.8) for a proper choice of K [A15].

Since T^{K+1} is a pseudo-contraction mapping, its convergence is geometric on the prescribed set. Since $T^k J$ falls in the neighborhood of J^* after a finite number of iterations and thereafter the convergence is geometric, the above theorem gives a partial answer to the open question of the rate of convergence. Other important open questions are about the formulation of bounds on the distance $\|J^* - T^k J\|$ in terms of the *residual* $\|TJ - J\|$, and bounds on the suboptimality of greedy policies. Of course, some trivial bounds like $\|J_\mu - J^*\| \le \|J_\mu - J\|$ always hold for $0 \le J \le J^*$. Value functions satisfying this last condition form an important class known as *admissible* value functions and will appear later in the algorithms. The rest of this section is devoted to the proofs of the theorems.

Proofs

It will be assumed without loss of generality that there is a unique goal state denoted by t (target). For the proof of Theorem 6, the following is needed

Lemma 8 *Assume a model like S1–S7 and A0–A2. Then, for $J_0 \overset{def}{\equiv} 0$ and any policy $\pi = (\mu_0, \mu_1, \ldots)$, $J_\pi = \limsup_{k\to\infty} T_{\mu_0} T_{\mu_1} \cdots T_{\mu_k} J_0$.*

Proof: A similar argument as the one in Lemma 3 works in this case. □

Lemma 9 *Assume a model like S1–S7 and A0–A2. Then, for any proper policy μ and vector J with $J(t) = 0$, $T^k_\mu J \to J_\mu$ uniformly as $k \to \infty$.*

Proof: Let J, J' be any two vectors with $J(t) = J'(t) = 0$ and $s_0 \in S$, then

$$\begin{aligned} &|(T^k_\mu J)(s_0) - (T^k_\mu J')(s_0)| \\ &\quad\le \left| \sum_{s_1 \in S} p(s_0, \mu(s_0), s_1)\left[(T^{k-1}_\mu J)(s_1) - (T^{k-1}_\mu J')(s_1)\right] \right| \end{aligned}$$

$$\begin{aligned}
&\le \sum_{s_1\in S} p(s_0,\mu(s_0),s_1)|(T_\mu^{k-1}J)(s_1)-(T_\mu^{k-1}J')(s_1)| \\
&\le \sum_{s_1\in S}\cdots\sum_{s_k\in S} p(s_0,\mu(s_0),s_1)\cdots p(s_{k-1},\mu(s_{k-1}),s_k)|J(s_k)-J'(s_k)| \\
&\le P_{\mu,s_0}(s_k\neq t)\,\|J-J'\|.
\end{aligned}$$

Thus, for $k=N_\mu$, the above computation shows $\|T_\mu^k J - T_\mu^k J'\| \le \rho_\mu \|J-J'\|$. Then, T^k is a contraction so it has a unique fix point that is J_μ. Therefore, it is needed to show the convergence for the whole sequence and not only for the subsequence $\{T_\mu^{kN_\mu}\}_k$. But clearly, $\|TJ-TJ'\|\le\|J-J'\|$ which implies $\|T^{k+m}J-T^{k+m}J'\|\le\|T^kJ-T^kJ'\|$ for any non-negative integers k and m. Thus, let $\{k_i\}_i$ be an increasing subsequence of k, and let $\{k_{i(j)}\}_j$ be a further subsequence such that $k_{i(j+1)}\ge k_{i(j)}+N_\mu$. Therefore,

$$\|T_\mu^{k_{i(j+1)}}J-T_\mu^{k_{i(j+1)}}J'\| \;\le\; \rho_\mu\,\|T_\mu^{k_{i(j)}}J-T_\mu^{k_{i(j)}}J'\| \;\le\; \rho_\mu^j\,\|T_\mu^{k_{i(1)}}J-T_\mu^{k_{i(1)}}J'\|,$$

and the theorem follows since the right-hand side goes to 0 and the subsequence was arbitrary. □

Proof of Theorem 6: For the first claim, let $J_0 \overset{\text{def}}{\equiv} 0$ and use the facts $T^kJ_0 \le T_\mu^kJ_0$ and $J_0\le TJ_0$ with the monotonicity of T and T_μ to conclude the important fact

$$T^kJ_0 \;\le\; T^{k+1}J_0 \;\le\; \lim_{k\to\infty}T_\mu^kJ_0 \;=\; J_\mu\,;$$

i.e. that $\{T^kJ_0\}_k$ is an increasing sequence within a compact set. Therefore, it has a limit point J^* [A3] that satisfies, by the continuity of the operator T, $TJ^*=T(\lim_{k\to\infty}T^kJ_0)=\lim_{k\to\infty}T^{k+1}J_0=J^*$.

For uniqueness, suppose that J^{**} is another fixed point and let μ^{**} and μ^* be the greedy policies with respect to J^{**} and J^* respectively; i.e. $T_{\mu^*}J^*=TJ^*$ and similarly for μ^{**}. Note that μ^* is proper since $T_{\mu^*}^kJ^*=T^kJ^*=J^*\le J_\mu$ (that is, it is uniformly bounded). Then, $J^{**}=TJ^{**}\le T_{\mu^*}J^{**}$ and Lemma 9 implies that $J^{**}\le J^*$. Therefore, J^{**} is in a compact set and a similar computation using Lemma 8 shows that μ^{**} is a proper policy. Thus, using Lemma 9 again, $J^*=TJ^*\le T_{\mu^{**}}J^*$ implies that $J^*\le J^{**}$ and both fixed points are equal.

For the optimality of μ^*, observe that the same argument that established the optimality of μ^* in the proof of Theorem 4 can be applied in this case.

To finish the proof, it remains to show the last part of *(iii)*. First, assume that $J\le TJ\le J^*$. Then, T^kJ is a bounded and increasing sequence so it converges to J^*. By the same token, the convergence also holds for the case $J^*\le TJ\le J$. Now, in the general case for J in a compact set, choose $\delta>0$ large enough so that $J_0+E_{-\delta}\le J\le J_\mu+E_\delta$ where E_δ is the value function satisfying $E_\delta(s)=\delta$ for $s\neq t$ and $E_\delta(t)=0$. Clearly,

$$\begin{aligned}
(T(J_0+E_{-\delta}))(s) &= (TJ_0)(s)-\delta \;\ge\; J_0(s)-\delta \quad \text{and} \\
(T(J_\mu+E_\delta))(s) &= (TJ_\mu)(s)+\delta \;\le\; J_\mu(s)+\delta,
\end{aligned}$$

since $TJ_\mu\le T_\mu J_\mu=J_\mu$. Therefore, $J_0+E_{-\delta}\le T(J_0+E_{-\delta})\le J^*$ and $J^*\le T(J_\mu+E_\delta)\le J_\mu+E_\delta$. Then, applying T^k and taking limits,

$$\begin{aligned}
J^* \;=\; \liminf_{k\to\infty}\;T^k(J_0+E_{-\delta}) \;&\le\; \liminf_{k\to\infty}\;T^kJ \\
&\le\; \limsup_{k\to\infty}\;T^kJ \;\le\; \limsup_{k\to\infty}\;T^k(J_\mu+E_\delta) \;=\; J^*.
\end{aligned}$$

Thus, equality holds throughout and $\lim_{k\to\infty}T^kJ=J^*$. □

For the proof of Theorem 7, it will be assumed that

A3. There are no self-loops in the state transition diagram except for goal states; i.e. $p(s,a,s)=0$ for all $s\in S\setminus S_G$ and $a\in A(s)$.

The first step towards the proof of Theorem 7 is to define the following stochastic process that, for a vector J with $J(t) = 0$, starts at state s_0 and vector J_0 given by

1. At time 0, let $X_0 = s_0$ and $J_0 = T^{n+1}J$.
2. At each time k, choose action $A_k \in A(X_k)$ greedy with respect to $T^{n-k-1}J$.
3. Jump to state $X_{k+1} = s$ with probability $p(X_k, A_k, s)$.
4. Update vector as $J_{k+1}(s) \overset{\text{def}}{=} [\![s = X_k]\!](T^{n-k}J)(s) + [\![s \neq X_k]\!]J_k(s)$.

where n is an integer whose value will be determined later. Denote with $\hat{P}_{s_0,J_0}$ the probability measure associated with the stochastic process and with $\hat{E}_{s_0,J_0}$ the expectation with respect to this measure.

Assume for the moment the validity of the following implication where $K < n$ is a non-negative integer and $0 \leq \alpha < 1$:

$$\frac{\|J^* - J\|_1 + \|J^*\|}{\underline{g}(K+1)} + \frac{\|J^* - J\|}{\underline{g}} \;\leq\; \alpha \quad\Longrightarrow\quad \max_{s \in S} \hat{P}_{s,J_0}(X_K \neq t) \;\leq\; \alpha \tag{3.9}$$

for cost vectors J satisfying $0 \leq J \leq TJ$. Then, Theorem 7 holds since for $s_0 \in S$,

$$\begin{aligned}
&|J^*(s_0) - (T^nJ)(s_0)| \\
&\quad= J^*(s_0) - (T^nJ)(s_0) \\
&\quad= \min_{a \in A(s_0)} \Big(g(s_0,a) + \sum_{s_1 \neq t} p(s_0,a,s_1)J^*(s_1) \Big) - \\
&\qquad\qquad \min_{a \in A(s_0)} \Big(g(s_0,a) + \sum_{s_1 \neq t} p(s_0,a,s_1)(T^{n-1}J)(s_1) \Big) \\
&\quad\leq \sum_{s_1 \neq t} p(s_0,a_0,s_1)\big[J^*(s_1) - (T^{n-1}J)(s_1)\big] \\
&\quad\leq \sum_{s_1 \neq t} p(s_0,a_0,s_1) \sum_{s_2 \neq t} p(s_1,a_1,s_2)\big[J^*(s_2) - (T^{n-2}J)(s_2)\big] \\
&\quad\leq \sum_{s_1 \neq t} \cdots \sum_{s_k \neq t} p(s_0,a_0,s_1)\cdots p(s_k,a_{k-1},s_k)\big[J^*(s_k) - T^{n-k-1}J(s_k)\big] \\
&\quad\leq \hat{P}_{s_0,J_0}(X_k \neq t)\,\|J^* - T^{n-k-1}J\|
\end{aligned}$$

where $a_k \in A(s_k)$ are the greedy actions with respect to $T^{n-k-1}J$. Thus, for $n = K+1$,

$$\|J^* - T^{K+1}J\| \;\leq\; \max_{s_0 \neq t} \hat{P}_{s_0,J_0}(X_K \neq t)\,\|J^* - J\| \;\leq\; \alpha\|J^* - J\|$$

and Theorem 7 holds. If (3.9) holds for some J, then it also holds for T^kJ. Therefore, it is enough to show the validity of (3.9).

Lemma 10 *For all $0 \leq J \leq TJ$ and $k \geq 0$, $J_k \geq T^{n-k+1}J \geq T^{n-k-1}J$ where n is the integer associated with the process J_k.*

Proof: The second inequality is trivial by the assumption $J \leq TJ$. The proof of the first is by induction. The base case $k = 0$ is obvious since $J_0 = T^{n+1}J$. Assume the validity for k, then

$$\begin{aligned}
J_{k+1}(s) &= [\![s = X_k]\!](T^{n-k}J)(s) + [\![s \neq X_k]\!]J_k(s) \\
&\geq [\![s = X_k]\!](T^{n-k}J)(s) + [\![s \neq X_k]\!](T^{n-k+1}J)(s)
\end{aligned}$$

$$\begin{aligned} &\geq [\![s = X_k]\!](T^{n-k}J)(s) + [\![s \neq X_k]\!](T^{n-k}J)(s) \\ &= T^{n-k}J\,. \end{aligned}$$

□

Theorem 11 *Assume a model like S1–S7 and A0–A3. Then, for all J such that $0 \leq J \leq TJ$,*

$$\hat{P}_{s_0,J_0}([\![X_k \neq t]\!] \leq \frac{\|J^*\| + \|J^* - J\|_1}{\underline{g}(k+1)} + \frac{\|J^* - J\|}{\underline{g}}\,.$$

Proof: Consider the sequence $(W_k)_{k\geq 0}$ defined as

$$\begin{aligned} W_0 &\overset{\text{def}}{=} \sum_{s\in S}\big[J^*(s) - J_0(s)\big] - J_0(s_0), \\ W_{k+1} &\overset{\text{def}}{=} \sum_{s\in S}\big[J_{k+1}(s) - J(s)\big] - J_k(X_{k+1})\,. \end{aligned}$$

Let $\mathcal{F}_k = \{X_0, A_0, \dots, X_k, A_k\}$ be the 'random history' of the process until time k (also known as filtration). Then,

$$\begin{aligned} W_{k+1} &= \sum_{s\in S}\big[J_{k+1}(s) - J(s)\big] - J_k(X_{k+1}) \\ &= \sum_{s\neq X_k}\big[J_{k+1}(s) - J(s)\big] + J_{k+1}(X_k) - J(X_k) - J_k(X_{k+1}) \\ &= \sum_{s\neq X_k}\big[J_k(s) - J(s)\big] + J_{k+1}(X_k) - J(X_k) - J_k(X_{k+1}) \\ &= \sum_{s\in S}\big[J_k(s) - J(s)\big] - J_k(X_k) + J_{k+1}(X_k) - J_k(X_{k+1}) \\ &= \sum_{s\in S}\big[J_k(s) - J(s)\big] - J_{k-1}(X_k) + J_{k+1}(X_k) - J_k(X_{k+1}) \\ &= W_k + (T^{n-k}J)(X_k) - J_k(X_{k+1}) \\ &= W_k + g(X_k, A_k) + \sum_{s\in S} p(X_k, A_k, s)(T^{n-k-1}J)(s) - J_k(X_{k+1}) \\ &= W_k + g(X_k, A_k) + \hat{E}_{s_0,J_0}[(T^{n-k-1}J)(X_{k+1})|\mathcal{F}_k] - J_k(X_{k+1}) \end{aligned}$$

using the facts $J_{k+1}(s) = J_k(s)$ for $s \neq X_k$, and $J_k(X_k) = J_{k-1}(X_k)$ since $X_k \neq X_{k-1}$ by assumption A3. Taking expectations with respect to $\mathcal{F}_k$,

$$\begin{aligned} \hat{E}_{s_0,J_0}[W_{k+1}|\mathcal{F}_k] &= W_k + g(X_k, A_k) + \hat{E}_{s_0,J_0}[(T^{n-k-1}J)(X_{k+1}) - J_k(X_{k+1})|\mathcal{F}_k] \\ &\geq W_k + g(X_k, A_k) - \|T^{k+2}J - J\| \\ &\geq W_k + \underline{g}[\![X_k \neq t]\!] - \|J^* - J\|\,. \end{aligned}$$

The first inequality by Lemma 10. Unfolding the first term on the right-hand side by integrating with respect to previous $\mathcal{F}_k$'s, one obtains

$$\hat{E}_{s_0,J_0}[W_{k+1}|\mathcal{F}_{k-i}] \geq W_{k-i} + \underline{g}\sum_{j=0}^{i}\hat{P}_{s_0,J_0}(X_{k-j} \neq t|\mathcal{F}_{k-i}) - (k-i+1)\|J^* - J\|$$

Now, go all the way down to W_0 to get

$$\hat{E}_{s_0,J_0}[W_{k+1}] \geq W_0 + \underline{g}\sum_{j=0}^{k}\hat{P}_{s_0,J_0}(X_j \neq t) - (k+1)\|J^* - J\|$$

$$\geq W_0 + \underline{g}(k+1)\hat{P}_{s_0,J_0}(X_k \neq t) - (k+1)\|J^* - J\|$$

since the monotonicity of the events $\{X_k \neq t\} \supseteq \{X_{k+1} \neq t\}$. The result is then a consequence of the following bounds:

$$\begin{aligned} W_0 &= \|J_0 - J\|_1 - J_0(s_0) \geq \|J_0 - J\|_1 - \|J^*\|, \\ W_{k+1} &= \|J_{k+1} - J\|_1 - J_k(X_{k+1}) \leq \|J^* - J\|_1 . \end{aligned}$$

□

3.3 Partially Observable MDPs

In cases where the assumption A0 is not satisfied, the agent or controller does not have complete knowledge about the current state of the system so its decisions need to be based on *estimations*. These estimations are computed from the history of past actions and *feedback* received. Thus, the model adds a new abstract ingredient, called observations, that are used to model the different feedback signals generated by the system. The feedback signal ranges in a spectrum from complete to null feedback. In the first case when the feedback signal is complete, it unequivocally identifies the state of the system and so the assumption A0 is satisfied and the model reduces to a standard MDP. On the other extreme, when the feedback signal is constantly *null*, the controller receives no information whatsoever and so, its decisions need to work for any possible trajectory of the system. The middle ground corresponds to a feedback signal carrying partial information about the state, and the controller needs to use this information in the best possible way. This general model is known as Partially Observable Markov Decision Process or POMDP; some standard references are [5, 186, 137] while an introduction from the perspective of planning and AI can be found in [43, 113, 26].

3.3.1 Non-Deterministic Systems

The non-deterministic POMDP model consists of the following ingredients [26]:

NP1. A finite state space S,

NP2. finite set of actions $A(s) \subseteq A$ for each state $s \in S$,

NP3. a set S_{init} of possible initial states,

NP4. a set of *goal* states $S_G \subseteq S$,

NP5. a *non-deterministic* transition function $F(s, a)$ for $s \in S$ and $a \in A(s)$ that is equal to the set of possible successor states after applying action a in s,

NP6. a *feedback signal* (also known as *sensor model*) $O(s, a) \subseteq O$ for all states s and actions $a \in A(s)$ that is equal to the *finite* set of possible feedbacks received when *entering* the state s after applying the action a, and

NP7. action costs $g(s, a)$ associated to each $s \in S$ and $a \in A(s)$.

An example of this type of system is the game of Mastermind given in the introduction. As it was shown in Ch. 2, the actions in this case do not modify the state of the system but only return information about it. Thus, this is an example of a problem in which the sensor model $O(s, a)$ depends on both the current state of the system and the applied action.

The case of complete feedback is obtained from NP1–NP7 by setting $O(s, a) = \{s\}$. Thus, every time the system enters state s the controller receives a feedback signal $= s$ telling the current state. On the other hand, the case of a null feedback corresponds to a function $O(s, a) \overset{\text{def}}{\equiv}$ const. for all states and actions. This latter case, which is considerably simpler than the former and known as a *conformant* planning problem [185] shall be considered first before the more general case.

Instead of defining the transition system for this model, which is non Markovian since the actions might depend on all previous states, the model NP1–NP7 is transformed into an associated MDP in *belief space* called the *belief*-MDP (see also [94]).

The term *belief state* refers to a *set of states*.[3] A belief state b stands for the states that the agent executing the policy deems possible at one point. The initial belief state b_{init} is given by the set S_{init} of possible initial states, and if b expresses the belief state prior to performing action a, the belief state b_a describing the possible states in the next situation is

$$b_a \overset{\text{def}}{=} \{s \in S \mid \exists(s' \in b)(s \in F(s', a))\} \tag{3.10}$$

The set $A(b)$ of actions that can be *safely* applied in a belief state b are the actions a that can be applied in *any* state that is possible according to b:

$$A(b) \overset{\text{def}}{=} \{a \mid \forall(s \in b)(a \in A(s))\} \tag{3.11}$$

The task in these models is to find a sequence of applicable actions that maps the initial belief state b_{init} into a *final belief state* containing only *goal states*. That is, the set B_G of *target* beliefs is given by

$$B_G \overset{\text{def}}{=} \{b \mid \forall(s \in b)(s \in S_G)\}. \tag{3.12}$$

Finally, the cost of applying an action in a belief state is given by

$$g(b, a) \overset{\text{def}}{=} \max_{s \in b} g(s, a)\,. \tag{3.13}$$

Provided with these definitions, a conformant version of a model like NP1–NP7 can be formulated as the following *Deterministic Decision Problem* in belief space.

DB1. The *finite* state space B of *belief states* b over S,

DB2. an initial state given by a belief state $b_{init} \in B$,

DB3. goal states given by the target beliefs (3.12),

DB4. actions $A(b) \subseteq A$ applicable in b given by (3.11),

DB5. a dynamics in which every action $a \in A(b)$ deterministically maps b into b_a as in (3.10), and

DB6. action costs $g(b, a)$ given by (3.13).

For an example of the belief-MDP associated to a POMDP, consider the game of Mastermind given in Ch. 1 and Fig. 1.1 where the belief states and transitions between them are clearly shown.

The search methods that are used to solve deterministic decision problems over *states* can then be used to solve deterministic decision problems over *belief states*.

In the presence of feedback, observations provide *information* about the current state but do not necessarily identify it since different states can produce the same observation; this is often called ‘perceptual aliasing’ [49]. On the other hand, after applying the action a and gathering observation o, it is known that the current state of the system is *not* s for $o \notin O(s, a)$.

Prior to performing an action a in a belief state b, there is a set of possible observations that may result from the execution of a, as the observations depend on the (unobservable) state s that results. This set of possible observations, denoted by $O(b, a)$, is

$$O(b, a) \overset{\text{def}}{=} \cup\{O(s, a) \mid s \in b\} \tag{3.14}$$

After a is done one of these observations o must obtain, allowing the agent to exclude from b_a the states that are not compatible with o. The resulting belief, called b_a^o, is then

$$b_a^o \overset{\text{def}}{=} \{s \in b_a \mid \exists o(o \in O(s, a))\} \tag{3.15}$$

Since the observation o that is obtained cannot be predicted, the effect of actions on beliefs is *non-deterministic* and thus an action a in a belief state b can lead to *any* belief state b_a^o for $o \in O(b, a)$.

Non-deterministic POMDPs can thus be modeled as *non-deterministic* MDPs over belief space given by

[3]A terminology that is borrowed from the logics of knowledge [79] and POMDPs [113].

NB1. The finite state space B of belief states b over S,

NB2. an initial state given by a belief state $b_{init} \in B$,

NB3. goal states given by the target beliefs (3.12),

NB4. actions $A(b) \subseteq A$ applicable in b given by (3.11),

NB5. sets of possible observations $O(b, a) \subseteq O$ after doing a in b given by (3.14),

NB6. a dynamics in which every action $a \in A(b)$ non-deterministically maps b into b_a^o for the different $o \in O(b, a)$ given by (3.15), and

NB7. action costs $g(b, a)$ given by (3.13).

The conditions that guarantee existence and convergence for non-deterministic MDPs apply in this case; thus, for example, if there exists a proper policy (in belief space) and all costs are positive (except for goal beliefs), then

(i) There exists an optimal policy in belief space,

(ii) there exists an optimal policy that is stationary, and

(iii) the optimal value function is the unique solution to the Bellman's equation in belief space.

And similarly for other properties of non-deterministic MDPs. However, in the stochastic case the model becomes much more complex as it is shown next.

3.3.2 Stochastic Systems

The most general and complex models that are considered in this dissertation correspond to partially observable systems with both stochastic transitions and feedback. In this case, the non-deterministic transition function $F(s, a)$ is replaced by transition probabilities $p(s, a, s')$ and the sensor model $O(s, a)$ is replaced by sensor probabilities $q(s, a, o)$ with a similar interpretation as before.

In this context, belief states are no longer subsets of states but *probability distributions* over states. The probability that s is the current state of the system is expressed by $x(s)$ (from now on, probabilistic belief states will be denoted with $x, y, \ldots$ whereas $b, b', \ldots$ will be used for subsets of states). The effect of actions and observations on beliefs are captured by equations analogous to (3.10)–(3.15) that are derived from the transition and feedback probabilities using Bayes' rule:

$$x_a(s) \overset{\text{def}}{=} \sum_{s' \in S} p(s', a, s)\, x(s'), \tag{3.16}$$

$$x_a^o(s) \overset{\text{def}}{=} q(s, a, o) x_a(s) / q(x, a, o)\,; \quad \text{for } q(x, a, o) \neq 0. \tag{3.17}$$

Here $q(x, a, o)$ stands for a normalizing constant equal to the probability of receiving o after doing action a in x, which is given by

$$q(x, a, o) \overset{\text{def}}{=} \sum_{s \in S} q(s, a, o)\, x_a(s), \tag{3.18}$$

and the sets of possible observations obtained is $O(x, a) \overset{\text{def}}{=} \{o : q(x, a, o) > 0\}$. As before, the target belief states are the ones that make the goal states S_G certain, while the set $A(x)$ of applicable actions are those that are applicable in all states that are possible according to x. However, the action costs in this case do not correspond to worst-case costs but expected costs as defined by

$$g(x, a) \overset{\text{def}}{=} \sum_{s \in S} g(s, a)\, x(s) \tag{3.19}$$

The problem then becomes a *fully observable stochastic decision process* over the belief space generated by probability distributions over states given by

SB1. A *infinite* state space B of probability distribution over S,

SB2. a finite set of actions $A(x) \subseteq A$ for each belief $x \in B$,

SB3. an initial distribution $x_{init} \in B$ over states,

SB4. a set of goal beliefs $B_G \subseteq B$,

SB5. a dynamics where actions and observations map belief x into x_a^o with probability $q(x,a,o)$ according to (3.18),

SB6. action costs $g(x,a)$ given by (3.19), and

SB7. a discount factor $\alpha \in [0,1]$.

As the reader may have realized, the belief space B of the model SB1–SB7 is not only infinite but a *continuum*. This fact makes this model qualitatively different from previous ones and, indeed, much harder to analyze. However, in the discounted case (i.e. $\alpha < 1$) the corresponding Bellman's operators

$$(T_\mu J)(x) \overset{\text{def}}{=} g(x,\mu(x)) + \alpha \sum_{o \in O(x,\mu(x))} q(x,\mu(x),o) J(x^o_{\mu(x)}) \tag{3.20}$$

$$(TJ)(x) \overset{\text{def}}{=} \min_{a \in A(x)} \left(g(x,a) + \alpha \sum_{o \in O(x,a)} q(x,a,o) J(x^o_a) \right) \tag{3.21}$$

are contractions over B with respect to the supremum (max) norm. Then, a proof identically to that of Theorem 4 gives

Theorem 12 (Sondik [186]) *Assume the* MDP *model defined by SB1–SB7. If $\alpha < 1$, then (i) there exists an optimal policy π^*, (ii) there exists an optimal policy μ^* that is stationary (which is the greedy policy with respect to J^*), and (iii) the optimal value function J^* is the unique solution to the Bellman's equations $TJ = J$.*

Similarly, most of the results for MDPs remain valid in this case.

In the rest if this section, we present new results about topological properties of J^* for the case of POMDPs. More specifically, it will shown that when $\alpha < 1$, then the optimal value function is a uniformly continuous map [A17] from belief states to reals under a suitable metric [A5], whereas it is discontinuous in the metric induced by the supremum norm. The following is an extension of the work in [23].

These results enable us to develop an optimal algorithm for POMDPs based on discretizations of the belief space (see Ch. 7). This is an important result since it is the first optimal algorithm based on discretizations. Moreover, until know all optimal algorithms for POMDPs are based on the so-called Sondik's representation theorem [186] (see Ch. 7).

Consider the metric ρ over belief states given by

$$\rho(x,y) \overset{\text{def}}{=} \begin{cases} 2 & \text{if } x \text{ and } y \text{ have different support,} \\ \sum_{s \in S} |x(s) - y(s)| & \text{otherwise.} \end{cases} \tag{3.22}$$

Here, different support means that there exists a state s such that either $x(s) = 0$ and $y(s) > 0$, or $x(s) > 0$ and $y(s) = 0$; equivalently there is s such that $x(s)y(s) = 0$ and $x(s) + y(s) > 0$. Throughout the rest of this section, ρ will denote this distance and σ the distance induced by the supremum norm; i.e. $\sigma(x,y) = \max_{s \in S} |x(s) - y(s)|$. The following result establishes that ρ is indeed a metric, and gives some of its properties.

Theorem 13 *ρ is a distance metric over the belief space B. For all $x, y \in B$ with identical support, $a \in A(x)$ and $o \in O(x,a)$:*

(i) $A(x) = A(y)$ *and* $O(x,a) = O(y,a)$,

(ii) $\rho(x_a, y_a) \le \rho(x,y)$,

(iii) $|q(x,a,o) - q(y,a,o)| \le \sum_{s \in S} q(s,a,o) |x_a(s) - y_a(s)|$, *and*

(iv) $\rho(x_a^o, y_a^o) \leq \frac{2}{q_x \wedge q_y} \sum_{s \in S} q(s,a,o)|x_a(s) - y_a(s)|$

where $q_x = q(x,a,o)$, $q_y = q(y,a,o)$ *and* $a \wedge b = \min\{a,b\}$.

The main theorem of this section is the following.

Theorem 14 *Let* $\overline{g} \overset{def}{=} \max_{s \in S} \max_{a \in A(s)} |g(s,a)|$ *and assume a model given by SB1–SB7 with* $\alpha < 1$. *Then, the optimal cost function* J^* *is a* uniform continuous *map from* B *into* $\mathbb{R}$. *Indeed, for any reals* γ *and* ϵ *chosen such that*

$$\epsilon + \alpha(2\epsilon|O|)^\gamma - \epsilon^\gamma \; \leq \; 0, \tag{3.23}$$

then

$$|J^*(x) - J^*(y)| \; \leq \; \frac{\overline{g}}{1-\alpha} \, \rho(x,y)^\gamma$$

for all $x, y \in B$ *such that* $\rho(x,y) \leq \epsilon$.

Remarks: The continuity of J^* with respect to the topology induced by ρ [A16] follows easily from the fact that J^* is the limit of a uniform convergent sequence of continuous functions [A13]. The fact that it is uniform continuous is more interesting since the metric space (B, ρ) is not compact nor complete [A7,A9]. The theorem goes further by giving a bound on the modulus of continuity [A17].

The following result gives a possible choice for the parameters γ and ϵ needed in the theorem.

Theorem 15 *Fix* $\alpha < 1$ *and choose* $\alpha < \beta < 1$. *If* γ *and* ϵ *are given by* $\gamma \overset{def}{=} (\log \beta - \log \alpha)/\log(2|O|)$ *and* $\epsilon \overset{def}{=} (1-\beta)^{1/1-\gamma}$, *then the inequality* (3.23) *holds for any non-empty set* O.

To illustrate the parameters given by the above theorem, choose $\beta = 0.99$ and assume $|O| = 4$. Fig. 3.1 shows the plot for the parameters γ and ϵ as given in Theorem 15 for different values of the discount factor α. For a problem with discount factor 0.95, for example, it is enough to choose $\gamma = 0.01983$ and $\epsilon = 0.00911$.

As a final remark, a result like Theorem 14 is not possible for the topology generated by the metric σ in general POMDP problems as the following example shows.

Example. Let $S = \{s_1, s_2\}$ and consider two actions a_1, a_2 such that $A(s_1) = \{a_1\}$, $A(s_2) = \{a_1, a_2\}$, $g(s_1, a_1) = 1$, $g(s_2, a_1) = g(s_2, a_2) = 0$, $p(s_2, a_2, s_2) = 1$ and the transition probabilities for a_1 are given by two parameters p_1, p_2 as shown in Fig. 3.2. Also, there is just one observation that is always observed with probability 1. Each belief state in this problem is of the form $(p, 1-p)$ so it can be represented by $p \in [0,1]$. When $\alpha < 1$, the corresponding POMDP is guaranteed to have a solution, and it is easy to check that J^* becomes

$$J^*(p) \; = \; \begin{cases} p + \alpha J^*(pp_1 + (1-p)(1-p_2)) & \text{if } p > 0, \\ 0 & \text{if } p = 0. \end{cases}$$

Note that $pp_1 + (1-p)(1-p_2) = 1 + p(p_1+p_2-1) - p_2$. Thus, if $p_1 + p_2 = 1$, then $J^*(1-p_2) = 1 - p_2 + \alpha J^*(1-p_2)$. Hence, if $p_1 + p_2 = 1$,

$$J^*(p) \; = \; \begin{cases} p + \alpha(1-p_2)/(1-\alpha) & \text{if } p > 0, \\ 0 & \text{if } p = 0. \end{cases}$$

Clearly, J^* is discontinuous at $p = 0$ with respect to the σ-topology. On the other hand, $p = 0$ is an isolated point in the ρ-topology, so J^* is continuous with respect to the latter. Also note that the jump can be made arbitrary large. □

It is not hard to see that this example crucially depends on the fact that the sets of applicable actions $A(s)$ are not identical. In fact, the following shows that an example like this is not possible in such cases.

Theorem 16 *Assume that* $A(s) = A(s')$ *for all states* $s, s' \in S$. *Then, the optimal value function is uniform continuous in the* σ*-topology.*

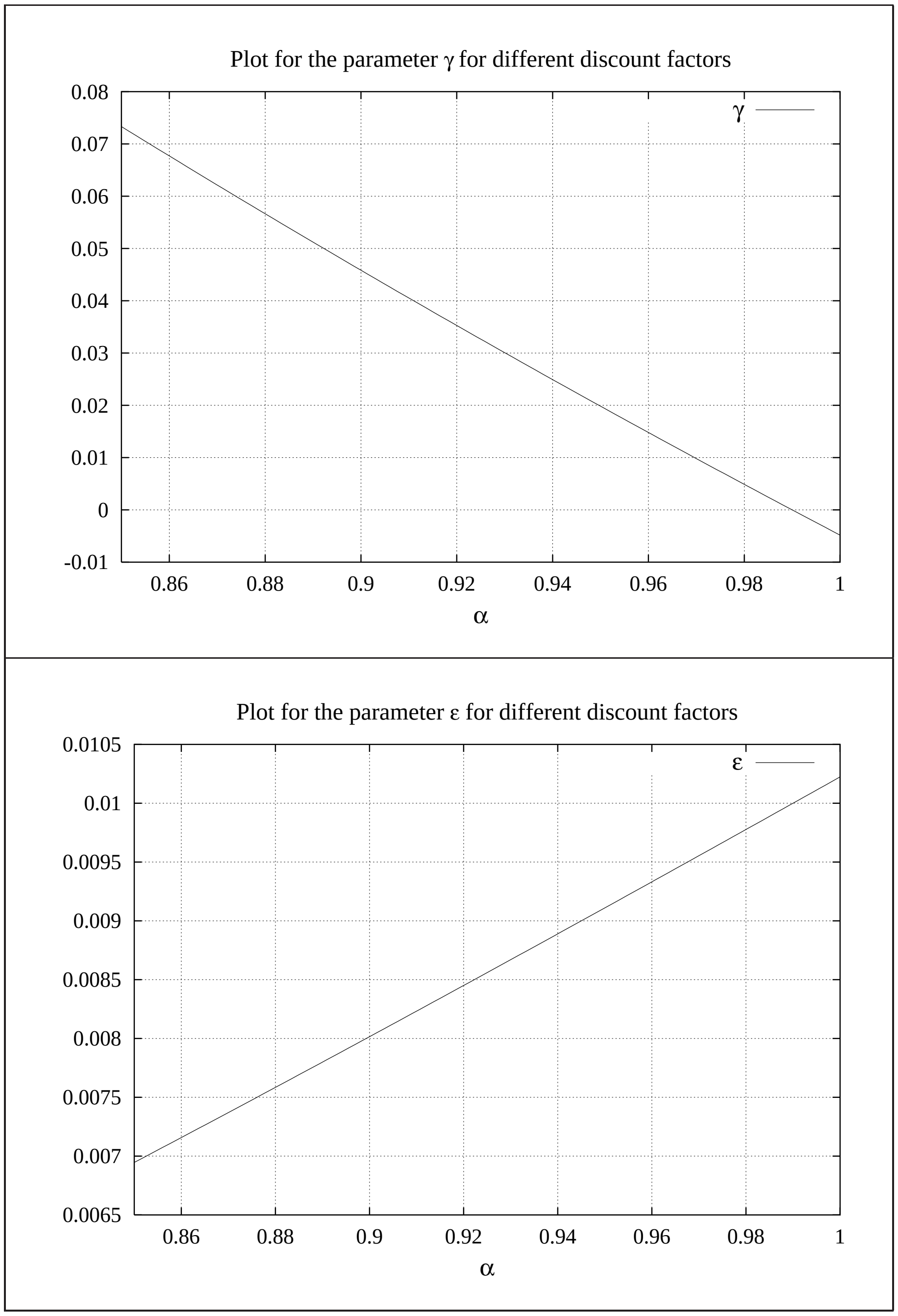


Figure 3.1: Plot for the parameters γ and ϵ as given in Theorem 15 for different values of the discount factor α.

$$\begin{tikzcd}
s_1 \arrow[loop above, "p_1"] \arrow[r, bend left, "1-p_1"] & s_2 \arrow[loop above, "p_2"] \arrow[l, bend left, "1-p_2"]
\end{tikzcd}$$

Figure 3.2: Transition probabilities for action a_1 in the example.

Proofs

Proof of Theorem 13: That ρ is a metric is trivial so it is left to the reader. Let $x, y \in B$ be such that $\rho(x,y) < 2$. Since x and y have identical support, it is obvious that $A(x) = A(y)$ and $O(x,a) = O(y,a)$. For (ii) and (iii),

$$\begin{aligned} \rho(x_a, y_a) &= \sum_{s\in S} \left| \sum_{s'\in S} p(s',a,s)\,(x(s') - y(s')) \right| \\ &\le \sum_{s,s'\in S} p(s',a,s)\,|x(s') - y(s')| \;=\; \rho(x,y), \end{aligned}$$

$$\begin{aligned} |q(x,a,o) - q(y,a,o)| &= \left| \sum_{s\in S} x_a(s) q(s,a,o) - y_a(s) q(s,a,o) \right| \\ &\le \sum_{s\in S} q(s,a,o)\,|x_a(s) - y_a(s)|. \end{aligned}$$

For (iv), let $q_x = q(x,a,o)$ and $q_y = q(y,a,o)$ and assume, without loss of generality, $q_x > q_y$. Then,

$$\begin{aligned} \rho(x_a^o, y_a^o) &= \sum_{s\in S} |x_a^o(s) - y_a^o(s)| \\ &= \sum_{s\in S} q(s,a,o) \left| \frac{x_a(s)}{q(x,a,o)} - \frac{y_a(s)}{q(y,a,o)} \right| \\ &= \frac{1}{q_x q_y} \sum_{s\in S} q(s,a,o)\,|q_y x_a(s) - q_x y_a(s)| \\ &\le \frac{1}{q_x q_y} \sum_{s\in S} q(s,a,o)\,\big(q_x\,|x_a(s) - y_a(s)| + |q_x - q_y| x_a(s)\big) \\ &= \frac{1}{q_x q_y} \left(q_x \sum_{s\in S} q(s,a,o)|x_a(s) - y_a(s)| + |q_x - q_y| \sum_{s\in S} q(s,a,o) x_a(s) \right) \\ &= \frac{1}{q_x q_y} \left(q_x \sum_{s\in S} q(s,a,o)\,|x_a(s) - y_a(s)| + |q_x - q_y| q_x \right) \\ &\le \frac{2}{q_x \wedge q_y} \sum_{s\in S} q(s,a,o)\,|x_a(s) - y_a(s)| \end{aligned}$$

as needed, where the last inequality follows from part (iii). □

Proof of Theorem 14: Let $x, y \in B$ be so that $\rho(x,y) < \epsilon$, thus $A(x) = A(y)$ and $O(x,a) = O(y,a)$. First, using induction it will be shown

$$|(T^k J_0)(x) - (T^k J_0)(y)| \;\le\; \frac{\overline{g}}{1-\alpha} \rho(x,y)^\gamma.$$

Indeed, for the base case,

$$\begin{aligned} |(TJ_0)(x) - (TJ_0)(y)| &= \Bigg| \min_{a\in A(x)} \left(g(x,a) \;+\; \alpha \sum_{o\in O(x,a)} q(x,a,o) J_0(x_a^o) \right) \\ &\qquad - \min_{a\in A(y)} \left(g(y,a) \;+\; \alpha \sum_{o\in O(y,a)} q(y,a,o) J_0(y_a^o) \right) \Bigg| \\ &\le |g(x,a) - g(y,a)| \end{aligned}$$

$$\leq\ \overline{g}\rho(x,y)\ \leq\ \frac{\overline{g}}{1-\alpha}\rho(x,y)^{\gamma}$$

since $\xi \leq \xi^{\gamma}$ for $0 \leq \xi \leq 1$. The control $a \in A(x)$ in the first inequality is the control that minimizes the second term, and it is assumed without loss of generality that the first term is larger than the second. The inductive step is

$$\begin{aligned}
&|(T^{k+1}J_0)(x) - (T^{k+1}J_0)(y)| \\
&\quad\leq\ |g(x,a) - g(y,a)| \\
&\qquad\qquad +\ \alpha\left|\sum_{o\in O(x,a)} q(x,a,o)(T^kJ_0)(x_a^o) - q(y,a,o)(T^kJ_0)(y_a^o)\right| \\
&\quad\leq\ \overline{g}\rho(x,y)\ +\ \alpha\sum_{o\in O(x,a)}\left|q_x(T^kJ_0)(x_a^o) - q_y(T^kJ_0)(y_a^o)\right| \\
&\quad\leq\ \overline{g}\rho(x,y) \\
&\qquad\qquad +\ \alpha\sum_{o\in O(x,a)}\Big[(q_x\wedge q_y)|(T^kJ_0)(x_a^o) - (T^kJ_0)(y_a^o)|\ +\ \|T^kJ_0\|\,|q_x - q_y|\Big] \\
&\quad\leq\ \overline{g}\rho(x,y)\ +\ \frac{\alpha\overline{g}\rho(x,y)}{1-\alpha}\ +\ \frac{\alpha\overline{g}}{1-\alpha}\sum_{o\in O(x,a)}(q_x\wedge q_y)\rho(x_a^o,y_a^o)^{\gamma} \\
&\quad=\ \frac{\overline{g}\rho(x,y)}{1-\alpha}\ +\ \frac{\alpha\overline{g}}{1-\alpha}\sum_{o\in O(x,a)}(q_x\wedge q_y)\,\rho(x_a^o,y_a^o)^{\gamma} \\
&\quad\leq\ \frac{\overline{g}\rho(x,y)}{1-\alpha}\ +\ \frac{\alpha\overline{g}}{1-\alpha}\,(2\rho(x,y))^{\gamma}\sum_{o\in O(x,a)}(q_x\wedge q_y)^{1-\gamma} \\
&\quad\leq\ \frac{\overline{g}\rho(x,y)}{1-\alpha}\ +\ \frac{\alpha\overline{g}}{1-\alpha}\,(2\rho(x,y))^{\gamma}\,|O|^{\gamma} \\
&\quad=\ \frac{\overline{g}}{1-\alpha}\big[\rho(x,y) + \alpha(2\rho(x,y)|O|)^{\gamma}\big] \\
&\quad\leq\ \frac{\overline{g}}{1-\alpha}\rho(x,y)^{\gamma}\,.
\end{aligned}$$

The fourth inequality follows using the inductive hypothesis and the fact

$$\begin{aligned}
\sum_{o\in O(x,a)}\|T^kJ_0\|\,|q_x - q_y|\ &\leq\ \frac{\overline{g}}{1-\alpha}\sum_{o\in O(x,a)}\sum_{s\in S} q(s,a,o)|x_a(s) - y_a(s)| \\
&=\ \frac{\overline{g}}{1-\alpha}\rho(x_a,y_a)\ \leq\ \frac{\overline{g}}{1-\alpha}\rho(x,y)\,.
\end{aligned}$$

The fifth by part (iv) of Theorem 13, and the sixth because $\sum_{o\in O(x,a)}(q_x\wedge q_y)^{1-\gamma} \leq |O(x,a)|/|O(x,a)|^{1-\gamma}$ since the left expression is maximized when all $q_x = q_y = 1/|O(x,a)|$. Hence,

$$\sum_{o\in O(x,a)}(q_x\wedge q_y)^{1-\gamma}\ \leq\ |O(x,a)|^{\gamma}\ \leq\ |O|^{\gamma}.$$

The last inequality is by the choice of the parameters γ and ϵ. Therefore, the first claim holds and taking limits

$$|J^*(x) - J^*(y)|\ =\ \lim_{k\to\infty}\left|(T^kJ_0)(x) - (T^kJ_0)(y)\right|\ \leq\ \frac{\overline{g}}{1-\alpha}\rho(x,y)^{\gamma}$$

as it was needed. □

Corollary 17 *In the case of $\alpha < 1/2$, the stronger inequality $|J^*(x) - J^*(y)| \leq \overline{g}\rho(x,y)/(1-\alpha)(1-2\alpha)$ is valid for all beliefs $x, y \in B$ such that $\rho(x,y) < 2$.*

Proof: Similar but using $\rho(x_a^o, y_a^o) \leq \frac{2}{q_x \wedge q_y} \sum_{s \in S} p(s,a,o)|x_a(s) - y_a(s)|$. □

Proof of Theorem 15:

$$\begin{aligned} \epsilon + \alpha(2\epsilon|O|)^\gamma - \epsilon^\gamma &\leq 0 \qquad \text{if and only if} \\ \epsilon^\gamma[\epsilon^{1-\gamma} + \alpha(2|O|)^\gamma - 1] &\leq 0 \qquad \text{if and only if} \\ \epsilon^{1-\gamma} + \alpha(2|O|)^\gamma &\leq 1. \end{aligned}$$

Now, separating variables, equate the second term in the last sum with β to obtain the expression for γ. Plug this value back into the last inequality to obtain $\epsilon^{1-\gamma} \leq 1 - \beta$. □

Proof of Theorem 16: Each function $T^k J$ is a continuous map in the σ-topology. Hence, the compactness of (B, σ) implies that $T^k J$ is uniform continuous [A17]. Now, use the fact that J^* is the uniform limit of a sequence [A13] of uniform continuous functions to conclude that J^* is a continuous function [A17]. The uniform continuity of J^* follows with a further application of compactness. □

3.3.3 Stochastic Shortest-Path Problems in Belief Space

The extra room left in the definitions of proper policies and Theorem 6 now becomes useful. Under *exactly* the same assumptions A1–A2 as before, the following theorem for stochastic shortest-path problems in belief space holds. To my knowledge, this is the first result about the convergence of value iteration for stochastic shortest-path problems in belief space.

Theorem 18 *Assume an undiscounted model SB1–SB7 and A1–A2 (with respect to the belief-*MDP*). Then, (i) there exists an optimal policy μ^* that is stationary, (ii) J^* is the unique solution to Bellman's equations, and (iii) for any vector J in a compact set with $J(x) = 0$ for goal beliefs $x \in B_G$,*

$$\lim_{k\to\infty} T_{\mu^*}^k J \;=\; J_{\mu^*} \;=\; J^* \;=\; \lim_{k\to\infty} T^k J$$

where the convergence in the left limit is uniform over B.

This result finishes the presentation of the relevant mathematical definitions and results. In the subsequent chapters, different algorithms for solving the Bellman's equations are given.

3.4 Summary and Notes

This chapter presents the mathematical models used to characterize the different planning tasks. The models are Deterministic Decision Processes, Markov Decision Processes in their non-deterministic and stochastic versions, and the Partially Observable Markov Decision Processes also in their non-deterministic and stochastic versions.

Bellman's Principle of Optimality that plays a fundamental role in the analysis of these models seems to appear explicitly for the first time in [12]. That reference also contains the important value iteration and the policy iteration algorithms.

Most of the definitions and results of this chapter are well-known and can be found in [16]. One exception is the definition of proper policies that is given for systems with infinite state spaces, yet it reduces to the standard definition for the finite case. All proofs given for already known facts are new and differ in technique from the proofs in standard textbooks like [164, 16].

The non-deterministic cases are not treated in standard books of Dynamic Programming and seem to be original from AI. The first general formulation of such models for system with uncertainty and partial information appears in [26].

An important new result is given in the statement and proof of Theorem 6. The difference with the statement in [16] is that no assumptions about the finiteness (and even countability) of the state space are made, and the proof in the latter reference cannot be extended to this case.

Theorem 7 and its consequences are new. The proof, which is based on a coupling argument, is a slightly change of the proof of Theorem 29 (Ch. 6) which was inspired by a related argument in [119].

The first treatment of POMDPs seems to be [5]. The metric ρ and Theorem 14 are from [23] as well as that the optimal value function for the case of equal action sets is a uniformly continuous mapping. This fact seems to have been known by different people, but a pointer to a proof cannot be found in the literature. The case of unequal action sets is much more difficult as shown in Example 1. The question whether the optimal value function is uniformly continuous or continuous in the undiscounted case is still open.

A treatment of decision processes with continuous state spaces can be found in [48], yet the assumptions made there are too strong to be applied to the belief-MDP generated by a POMDP.

Chapter 4

Representation Language

This chapter introduces a novel representation language for sequential decision tasks involving uncertainty and partial information. The language is a logic-based action language that extends the popular STRIPS language used in classical planning. The semantics of the language is given in terms of transition systems defined by the mathematical models, i.e. given a description of a planning task using the representation language, there is a precise mathematical model associated with such task.

Mathematical models provide a characterization of planning tasks and their solutions. These models however do not offer a convenient *language* for expressing planning problems. The language in that case is an *explicit* one made of abstract sets of states and actions, and transition probability matrices and cost vectors. Explicit specifications are only feasible for small problems, and in large problems, the state space and transition dynamics need to be represented *implicitly* using a logic-based action language comprised of state variables and action rules. Good action languages allow for *compact* representation of problems with large sets of states and actions.

Action languages are an instance of the general idea of high-level representation languages. The use of these languages are fecund in AI and can be found across a large number of fields where they are used for *knowledge representation and reasoning*. Example cases, besides planning, are the language of Bayesian networks for representing probabilistic information [156, 111], influence diagrams for decision-theoretic tasks [180], causal networks for causal reasoning [160, 188], causal rule systems for default reasoning [167, 89], factored probabilistic models for MDPs and POMDPs [36, 37], AMPL for linear and integer programs [84] and OPL for constraint satisfaction problems [198]. Indeed, there is a growing agreement in the community towards understanding AI as a blend of *formal models* + *representation languages* + *algorithms*.

The idea of high-level representation languages is common in planning [161, 145, 86] where they draw insight from recent theories of action [93, 179, 92, 168], and their origin can be traced back to the STRIPS language in the early 1970's [82].

Finally, description languages are not only used to compactly represent transition systems but also to automatically *extract* information about a given instance to speed up the algorithms; e.g. heuristic functions in the case of algorithms based on search, and propositional invariants in the case of algorithms based on deductive methods.

4.1 The STRIPS Language and Extensions

The STRIPS planning language is a simple language based on propositional logic for deterministic planning. The basic elements of the language are constant symbols $\mathcal{K}$ that represent objects, relational symbols $\mathcal{R}$ used to express relations between objects and action symbols $\mathcal{A}$ that represent actions.

To illustrate the STRIPS representation language and its extensions, a fairly simple domain known as the Blocks World will be used. We first define the standard version of the domain and then extend it gradually in order to cover issues as non-determinism, partial observability, etc.

The standard version of the blocks world domain, originally due to [82], consists of a set of square blocks over a table in which each block can be either resting on the table or on top of another block, and where each block either

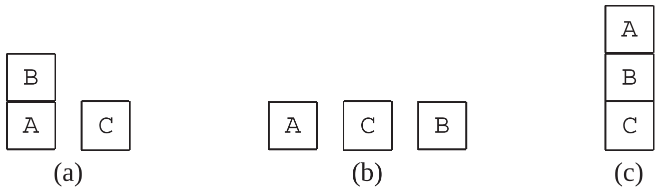


Figure 4.1: Simple situations in the blocks-world domain.

supports a single block or is clear. The configuration of blocks can be changed by a robot arm which can only picks clear blocks; that is, blocks that do not have any other block on top of them. The formulation given here only consider move operators whose arguments specify the block to move and the target location; yet other formulations are also possible.

In this version of the blocks world domain, the relational symbols are `on` (of arity 2) that describe whether a block rests on another block, and `on-table` and `clear` (of arity 1) that describe whether a block rests on the table or no other block rests on top of it. The situation shown in Fig. 4.1(a), for example, is fully described by the set of *propositions*:

{ `on-table(A), on(B,A), clear(B), on-table(C), clear(C)` }.

Thus, in STRIPS, each state of the system is described by the set of propositions that hold true in that state, and so there is a one-one correspondence between states and subsets of propositions. Note however that this relation is not onto since there are subsets of propositions that do not correspond to any feasible state; e.g.

{ `on-table(A), clear(A), on(B,A), on-table(C), clear(C)` }

since the block `A` cannot support block `B` and be clear at the same time. Thus, a set of propositions is called consistent or inconsistent depending on whether it describes a feasible state or not.

Each operator or action in STRIPS is characterized by means of three sets of propositions, originally called 'lists,' known as the *precondition*, *add* and *delete* lists. The precondition list is used to qualify the set of states where the action is applicable, while the add and delete lists are used to specify its effects. A given action with lists $(PREC, ADD, DEL)$ is by definition applicable in a state description S if $PREC \subseteq S$. Thus, as is standard in AI planning yet different from control theory and operations research, not every operator is applicable in every state. This is a substantial and powerful mechanism that helps the 'knowledge' engineer to build meaningful abstractions by leaving out unwanted details. For example, if the effects of an action that pick a block are undefined when there are other blocks on top of it, then such cases can be avoided with a simple precondition.

The effects of an action are defined, only when its preconditions are satisfied, by means of its add and delete lists as

$$S' := S \cup ADD \setminus DEL \tag{4.1}$$

where S' is the description of the state that results after the application of the action.[1] Thus, in the blocks-world domain, the following action can be applied to situation (a) to produce situation (b) shown in Fig. 4.1.

```
move-to-table(B,A)
    prec: on(B,A), clear(B)
     add: on-table(B), clear(A)
     del: on(B,A) .
```

The complete set of operators for the blocks-world domain formulation is generated with the following operator *schemata*:

[1]A common assumption in STRIPS is to require disjoint add and delete lists so that the union and difference set operators can be applied in any order.

```
move-to-table(?x,?y)
    prec: on(?x,?y), clear(?x)
     add: on-table(?x), clear(?y)
     del: on(?x,?y)

move-from-table(?x,?y)
    prec: on-table(?x), clear(?x), clear(?y)
     add: on(?x,?y)
     del: on-table(?x), clear(?y)

move-from-block(?x,?y,?z)
    prec: on(?x,?y), clear(?x), clear(?z)
     add: on(?x,?z), clear(?y)
     del: on(?x,?y), clear(?z) .
```

An operator schema is a template over object variables, denoted with symbols starting with `?`, that when *instantiated* generates different operators. Its use is standard practice since generally all operators can be partitioned into different classes with structural similarities within each class. The instantiation or *grounding* of operators follows by replacing each variable by a *different* object name. Thus, in the case of three blocks, the previous schemata will ground to 6 `move-to-table`'s, 6 `move-from-table`'s and 6 `move-from-block`'s for a total of 18 operators.

Note that the operators are defined so that consistency is preserved throughout their application. This crucial invariant is responsible for the soundness of the descriptions and qualifies STRIPS as a good description language.

The previous description faithfully describes the original STRIPS language, however STRIPS has been extended through the years in order to accommodate more expressive formalisms and planning models. The following section explores some of these extensions.

4.1.1 Types, Formulas and Conditional Effects

In complex domains, the relational symbols and operator structures depend on the nature or type of each object. In a colored version of the blocks-world domain there is an operator `paint(A,red,blue)` for changing the color of block `A` from red to blue (i.e. asserting `painted(A,blue)`), for example, the objects are divided into the two classes of blocks and colors, and the schema `paint(?x,?c1,?c2)` makes sense if the variable `?x` only refers to blocks, and the variables `?c1` and `?c2` only refer to colors. The fix to the language consists of dividing the objects into classes (or types) and qualifying schema variables with them; e.g. the paint operator translates into

```
paint(?x,?c1,?c2)
    types:?x - block, ?c1, ?c2 - color
     prec:painted(?x,?c1), clear(?x)
      add:painted(?x,?c2)
      del:painted(?x,?c1) .
```

Note that the variable `?c1` does not play an active role in the effects of the action but only provides a *handle* for properly removing `painted(?x,?c1)` from the state description in order to enforce consistency. This type of construct, that is often found in problem descriptions, is a consequence of the propositional nature of the language. Later, this restriction will be relaxed by using a more expressive language based on functional symbols.

Checking the precondition of an action in a state description S amounts to performing the subset inclusion test $PREC \subseteq S$. This is equivalent to the logical entailment test $s \models \varphi$ where s is the state described by S and φ is the *conjunctive* logical formula made from the propositions in $PREC$; e.g. $\varphi = \texttt{on(B,A)} \land \texttt{clear(B)}$ in the case of `move-to-table(B,A)`. Now, it is easy to obtain a useful generalization for the preconditions by considering general propositional formulas instead of just conjunctive ones. If the propositions obtained by grounding each action schema are called atoms, then the set $\mathcal{L}_P$ of propositional formulas is defined by

1. all atoms are in $\mathcal{L}_P$,

2. if φ and ψ are in $\mathcal{L}_P$ then so are $\neg\varphi$, $(\varphi \vee \psi)$ and $(\varphi \wedge \psi)$, and
3. nothing else is in $\mathcal{L}_P$.

Thus, from now on, the preconditions will be logical formulas from $\mathcal{L}_P$. Additionally, quantifier symbols will be used as *abbreviations* for describing complex propositional formulas; e.g. in the case of three colors `red`, `blue` and `yellow`, the abbreviation '`(exists (?c - color) painted(A,?c))`' stands for the disjunction

$$\texttt{painted(A,red)} \vee \texttt{painted(A,blue)} \vee \texttt{painted(A,yellow)}$$

and similarly for the conjunction generated by `forall`. The reader should bear in mind that the quantifications are just abbreviations and that the language is not a first-order language.[2]

So far, the use of quantification has been restricted for expressing action preconditions. To make a similar development for expressing effects, a syntactic modification is introduced. Instead of dividing the effects of an action into the add and delete lists, a single list, called the effect list, is used. The elements of the effect list are either atoms (propositions) or their negation, and the relation between the effect list and the add and delete lists is direct: the former is the add list conjoined with the list that results from negating all atoms in the delete list. In this new syntax, the schema for `move-to-table(?x,?y)` becomes

```
move-to-table(?x,?y)
    types:?x, ?y - block
     prec:on(?x,?y) ∧ clear(?x)
   effect:on-table(?x), clear(?y), ¬on(?x,?y) .
```

Now is possible to introduce the abbreviation '`foreach <var> do <effect> end-do`' for expressing multiple effects where `<var>` is a typed variable and `<effect>` is an effect list. For example, the operator `knock-table` which moves all blocks to the table can be defined as

```
knock-table
   effect:foreach (?x - block) do
             on-table(?x) ∧ clear(?x),
             foreach (?y - block) do ¬on(?x,?y) end-do
          end-do .
```

Some remarks are worth mentioning. First, note that the operator does not have parameters nor preconditions. Thus, it can be applied in any state and there is just one instance of it. Second, note how the foreach constructs are nested in order to achieve the intended effect, and finally note that the associated delete list contains all atoms of the form `on(?x,?y)` which cannot all be present in a given state description. This last fact causes no trouble since deleting a proposition which is not true in a state has no effect (a direct consequence of definition (4.1)).

The final extension of STRIPS considered in this section are *conditional* effects. As its name suggests, a conditional effect is an effect that is circumstantially applied in some states. Conditional effects, which are built with the syntax '`when <formula> do <effect> end-do`' where `<formula>` refers to an $\mathcal{L}_P$-formula and `<effect>` to an effect list, denote an effect that is only applied when the state satisfies the condition `<formula>`.

For an example of the use of conditional effects, consider a variations in which a block might be slippery depending on whether the block is wet or dry. Then, an attempt to move a wet block will fail leaving the configuration of blocks intact. In this variation of the domain, an operator `dry` over blocks is also available. The new version of `move-to-table` and the `dry` operator are as follows (similar modifications apply for the other move operators):

```
move-to-table(?x,?y)
    types:?x, ?y - block
     prec:on(?x,?y) ∧ clear(?x)
```

[2]However, an equivalent interpretation could be obtained by treating the language as a first-order language under the closed-world assumption semantics [166].

```
      effect:when ¬wet(?x) do
                on-table(?x), clear(?y), ¬on(?x,?y)
             end-do

   dry(?x)
       types:?x - block
        prec:clear(?x)
      effect:¬wet(?x) .
```

In the following section, a more general and expressive language for non-deterministic and stochastic effects, and partial observability is described. The new language is similar to STRIPS except that it is based on functional terms instead of propositional symbols.

4.2 The Planning Language

In order to cope with complex tasks, the STRIPS language will be extended in several directions. The first one consists of a departure from the propositional nature of the language to an instance of a first-order language with no quantification. This version of STRIPS, first defined by Geffner in [90], is called functional-STRIPS or f-STRIPS for short.

In f-STRIPS states are no longer described by sets of propositions but by *interpretations* over the set of *typed* terms (or algebra of typed terms) $\mathcal{T}$.[3]

Given a set of types (classes) $\mathcal{C}$, a finite set of constant symbols $\mathcal{K}$ and a finite set of *typed-functional* symbols $\mathcal{F}$, the algebra $\mathcal{T}$ is defined as

1. a constant symbol $c \in \mathcal{K}$ is a term with type $C \in \mathcal{C}$ if $C \ni c$,
2. if $f \in \mathcal{F}$ is an n-ary functional symbol of type C with formal arguments of types $C_1, \ldots, C_n$, and if $t_1, \ldots, t_n$ are terms with types $C_1, \ldots, C_n$ respectively, then $f(t_1, \ldots, t_n)$ is a term with type C,
3. nothing else is in $\mathcal{T}$.

Observe that the term algebra becomes infinite as soon as there are enough functional symbols to bootstrap a sustained generative process. So, in general, interpretations are infinite valuations, which is problematic in practical implementations. Therefore, in f-STRIPS, the set of allowable interpretations is restricted to the set of *interpretations that map terms into constant symbols*; i.e. the interpretations are functions $\mathcal{T} \to \mathcal{K}$. The reason to focus on this set of interpretations is that they are amenable to implementation as shown below; for clarity, only unary functional and relational symbols are considered.

Let ν denote one such interpretation and ν_f denote the subinterpretation given by ν to terms of the form $f(c)$ where c is a constant symbol. Plainly, $\nu(c) = c$ and $\nu(f(c)) = \nu_f(c)$ for all constant symbols c and functional symbols f, and inductively $\nu(f(t)) = \nu_f(c)$ where $c = \nu(t)$. Thus, the interpretation ν is characterized by the *finite* collection of finite-sized interpretations $\{\nu_f : f \in \mathcal{F}\}$.

Each ν_f can be represented as a table V_f indexed by the constant symbols while the relational symbols can be treated in a similar way by considering interpretations that map terms into boolean values. Therefore, the description S of a state s can be given by the finite collection of tables:

$$S = \{V_f : f \in \mathcal{F}\} \cup \{V_r : r \in \mathcal{R}\} \tag{4.2}$$

and the relation between logical entailment and the description S is

$$\begin{array}{lll} s \models f(c_1) = c_2 & \text{if and only if} & V_f[c_1] = c_2, \quad \text{and} \\ s \models r(c_1) & \text{if and only if} & V_r[c_1] = \text{true} \end{array}$$

[3]By interpretations is meant interpretations as the ones used in mathematical logic and model theory [45, 74].

for constant symbols c_1 and c_2, and the obvious extension to more complex terms; e.g. $s \models f(g(c_1)) = h(c_2)$ if and only if $V_f[V_g[c_1]] = V_h[c_2]$.

In presence of functional symbols, the set $\mathcal{L}_P$ needs to be extended in order to include all atoms made from the typed terms. The set of formulas $\mathcal{L}_F$ is then built as before from all atoms of the form $r(t)$ where r is a relational symbol and t is a term of appropriate type, or atoms of the form $(t_1 = t_2)$ for any two terms t_1 and t_2 of the same type; similar definitions are given in the case of general n-ary relational symbols.

A nice example of the functional language can be given on the previous extended instance of the blocks-world domain where blocks can have color and be wet or dry. An instance of this domain consisting of three blocks and three colors can be described with the following sets of constant, functional and relational symbols:

$$\begin{aligned}\mathcal{K} &= \{\, \texttt{A}, \texttt{B}, \texttt{C}, \texttt{table}, \texttt{red}, \texttt{blue}, \texttt{yellow} \,\},\\ \mathcal{F} &= \{\, \texttt{pos}, \texttt{color} \,\},\\ \mathcal{R} &= \{\, \texttt{clear}, \texttt{wet} \,\}\end{aligned}$$

with the types $\texttt{block} = \{\texttt{A}, \texttt{B}, \texttt{C}, \texttt{table}\}$ and $\texttt{color} = \{\texttt{red}, \texttt{blue}, \texttt{yellow}\}$. Of all these symbols, the only ones that require explanation are the functional symbols `pos` and `color` that denote the position and color of their arguments; e.g. `pos(A)` $=$ `table` and `color(A)` $=$ `red` if the block `A` rests on the table and is painted red. Here, the constant `table` is termed as a block in order to obtain the following operator that subsumes all previous versions of the move operator:[4]

```
move-block(?x,?y)
    types:?x, ?y - block
     prec:¬(?x = table) ∧ clear(?x) ∧ clear(?y)
   effect:when ¬wet(?x) do
             clear(pos(?x)), pos(?x) := ?y,
             when ¬(?y = table) do ¬clear(?y) end-do
          end-do .
```

Observe how the functional representation of the position allows the removal of one argument from the move operator, and that the first precondition is needed to enforce the consistency of the resulting states.

4.2.1 Non-Deterministic and Stochastic Effects

So far, each (grounded) action description consists of a precondition list and an effect list $(PREC, EFF)$. The action is applicable in a state description S whenever S satisfies $PREC$, and the result of the action is denoted as $S' = \text{result}(S, EFF)$.

For non-deterministic effects, the description of an action with multiple effects is a tuple $(PREC, EFF_1, EFF_2, \ldots, EFF_m)$ so that the set of possible resulting states after applying the action in description S is

$$\{\, S' : S' = \text{result}(S, EFF_j) \ \text{ for some } 1 \leq j \leq m \,\}.$$

The syntax for non-deterministic effects is shown with a new operator `weakmove` that either moves a block or drops it on the table:

```
weak-move(?x,?y)
    types:?x, ?y - block
     prec:¬(?x = table) ∧ clear(?x) ∧ clear(?y)
   effect:when ¬wet(?x) do
             clear(pos(?x)), pos(?x) := ?y,
             when ¬(?y = table) do ¬clear(?y) end-do
```

[4]To be precise, `pos(table)` needs to be defined since `table` $\in$ `block`. In our formulation however, `pos(table)` can be defined as any block since its denotation never change; a good choice is `pos(table)` $\overset{\text{def}}{=}$ `table` across all states.

```
             end-do
      effect:when ¬wet(?x) do
                clear(pos(?x)), pos(?x) := table
             end-do .
```

For stochastic effects, each individual effect EFF_j is tagged with a non-negative number p_j so that $\sum_j p_j = 1$. The syntax is similar to the one above except that each stochastic effect has syntax '`effect: <prob> <effect>`.'

4.2.2 Ramifications and Defined Predicates

In cases where a domain must enforce special constraints or invariants over states, that are otherwise difficult to express in the action rules, the use of *ramification rules* and explicit *logical invariants* are often useful.

A ramification rule is simply understood as a (action) rule that automatically *triggers* every time after the application of an action. Ramification rules are expressed using schemata with typed arguments and effects but no preconditions; the effects however need to be deterministic. In the presence of several ramification rules, they are applied in the static order given in the description.

Each explicit logical invariant, or simply invariant, is expressed by a formula from $\mathcal{L}_F$ that must hold in all states. Thus, after applying an action and all ramification rules, the states that do not satisfy the invariants are discarded. It is assumed that not all resulting states are discarded and invariants are only permitted for non-deterministic models. As with ramification rules, invariants are specified using templates parametrized with typed arguments. Appendix B contains a precise description of the syntax of the action language.

More interesting possibilities arise with the use of defined predicates. As its name suggest, a defined predicate defines a relational symbol $r(t)$ for terms t of a given type by means of a logical formula; e.g. in the blocks-world domain is natural to define

```
on-table(?x)
    types:?x - block
  defined:pos(?x) = table
```

and even to remove `clear` from $\mathcal{R}$ and define

```
clear(?x)
    types:?x - block
  defined:forall (?y - block) ¬(pos(?y) = ?x)) .
```

Yet a more expressive language is obtained by allowing simple *recursive* predicates. Thus, for example, the following is a valid definition

```
above(?x,?y)
    types:?x, ?y - block
  defined:(pos(?x) = ?y) ∨
          (exists (?z - block) (pos(?x) = ?z) ∧ above(?z,?y))
```

which properly defines the transitive closure of `pos`. Note that this is a genuine jump in expressive power since transitive closure cannot be defined in First-Order Logic (FOL) for an indeterminate number of objects. Only single recursion is allowed and hence mutually recursive predicates are not permitted.

4.2.3 Costs

All action costs are considered to be one by default, but different constant costs or costs that depend on the state are also supported. To specify a cost, the syntax '`cost: <cost>`' is used in the action's definition. The non-terminal `<cost>` can be substituted by:

1. a *positive* integer,
2. a *conditional cost* ‘`if <formula> <cost> <cost> fi`,’ or
3. a summation ‘`(sum <var> <cost>)`’ where `<var>` is a typed variable.

The complete syntax is given in Appendix B.

4.2.4 Partial Observability

Actions return information about the environment through observations. Observations can be either formulas, terms, or lists of observations. In the case of formulas, two different observations corresponding to the values true and false are possible, for terms the possible observations are given by the range of the functional term, and for lists of observations the set of potential observations is the cross product of the possibilities for each observation in the list.

Observations are specified with the entry ‘`obs: <obs>`’ in the action declaration. As in the case of stochastic effects, stochastic observations are specified with multiple observation entries tagged with probabilities; i.e. with multiple entries of the form ‘`obs: <prob> <obs>`.’ The following examples illustrate their use in a partially observable version of the blocks-world domain. In this variation, two sensing operators are available: `sense(?x)` that returns a single boolean value for the conjunction `clear(?x)` $\wedge$ `wet(?x)`, and `see(?x)` that returns a pair of boolean values for (`clear(?x),wet(?x)`):

```
sense(?x)
    types:?x - block
      obs:clear(?x) ∧ wet(?x)

see(?x)
    types:?x - block
      obs:clear(?x), wet(?x) .
```

Clearly, the `sense` action is more ‘informative’ than the `see` action since all states discriminated by the first are also discriminated by the second.

4.2.5 Initial and Goal Situations

We complete the description of the action language by giving the methods to specify the sets of goal and initial situations. The set of goal states is easy since it is specified with a logical formula using the syntax ‘`goal: <formula>`,’ i.e. a state is a goal state if it satisfies the given formula. In the case of partial observability, the goal condition ‘`full-knowledge`’ is also defined which specifies as goals not a particular state but those situations when the agent knows the exact state of the system.

The initial situation is more difficult since a way for specifying multiple initial situations is needed. For this, the language provides, besides standard deterministic effects, the statement ‘`<term> := <term-set>`’ for specifying that the first term can be any of the terms in the set, and the statement ‘`prune <formula>`’ that prunes all initial states that *do not* satisfy the formula. For example, the following initial and goal descriptions define a problem in which the initial state is any feasible configuration of the three blocks and the goal situation is shown in Fig. 4.1(c).

```
init:pos(A) := { A, B, C, table },
     pos(B) := { A, B, C, table },
     pos(C) := { A, B, C, table },
     prune (exists (?x - block) ¬above(?x,?x) ∨ (?x = table)),
     prune (exists (?x - block)
             (exists (?y - block)
               (exists (?z - block)
                 (?y = ?z) ∨ (?x = table) ∨ ¬(pos(?y) = ?x) ∨
                 ¬(pos(?z) = ?x)))),
objects = p0 p1 p2 - integer[0,1]

hit(?x)
    types: ?x - integer[0,1]
  defined: (?x = p0) ∨ (?x = p1) ∨ (?x = p2)

guess(?x0,?x1,?x2)
    types: ?x0, ?x1, ?x2 - integer[0,1]
      obs: (?x0 = p0) + (?x1 = p1) + (?x2 = p2),
           (hit ?x0) + (hit ?x1) + (hit ?x2)

init: p0 := { 0, 1 },
      p1 := { 0, 1 },
      p2 := { 0, 1 }

goal: full-knowledge
```

Figure 4.2: Description of the game of Mastermind with 3 pegs and 3 colors.

```
      pos(table) := table,
      color(A) := yellow,
      color(B) := blue,
      color(C) := red
goal:(pos(A) = B) ∧ (pos(B) = C) ∧ (pos(C) = table)
```

where the prune statements state that a block cannot be above itself and that no two different blocks can rest on the same block. This can be easily seen since they are logically equivalent to

$$\begin{aligned} \texttt{above(?x,?x)} &\implies \texttt{?x = table}, \text{ and} \\ \texttt{pos(?y) = ?x} \land \texttt{pos(?z) = ?x} &\implies \texttt{?x = table} \lor \texttt{?y = ?z}. \end{aligned}$$

4.3 Examples

Fig. 4.2 describes the instance of the game of Mastermind with 3 pegs and 2 colors given in the introduction. The description contains three objects `p0`, `p1` and `p2` for the color of the three pegs in the secret code, a defined predicate `hit(?x)` that is true if color `?x` appears in one of the pegs, an action schemeta `guess(?x0,?x1,?x2)` that returns the markers, a description of the initial situations with all possible 8 initial secret codes, and the goal condition of `full-knowledge`. Observe how succinct this representation is as opposed to one based on explicit models or even FOL.

The second example corresponds to the omelette problem also in the introduction. As said before, the problem consists of an agent that has a large supply of eggs and whose goal is to get n good eggs and no bad ones into one of two bowls. The eggs can be either good or rotten, and the agent can perform tasks as breaking an egg into a bowl, pouring the content of a bowl into another, and cleaning a bowl. The agent can also sense if a bowl contains at least one rotten egg by smelling it.

Fig. 4.3 contains a description of the omelette problem with 3 eggs. As was shown in the introduction, the solution to this problem involves a sequence of three 'loops' each one that guarantees obtaining a good egg in one of the two bowls.

The omelette problem corresponds to a stochastic planning problem with partial information. Observe how the use of the functional language allows for a short description of this problem that would be much larger in case of a propositional language. Also note the use of the predefined functional and relational symbols like '$+$' and '$<$' which

```
types     = bowl
relations = good-egg(), holding()
functions = goods :  bowl -> integer[0,3]
            bads :  bowl -> integer[0,3]
            number :  bowl -> integer[0,3]
objects   = small, large - bowl

ramification(?x)
    types: ?x - bowl
    ramif: number(?x) := goods(?x) + bads(?x)

grab-egg
     prec: ¬holding
   effect: 0.5 holding(), good-egg()
   effect: 0.5 holding(), ¬good-egg()

break-egg(?x)
    types: ?x - bowl
     prec: (number(?x) < 3) ∧ holding()
   effect: ¬holding(),
           when good-egg() do goods(?x) := 1 + goods(?x) end-do
           when ¬good-egg() do bads(?x) := 1 + bads(?x) end-do

pour-bowl(?x,?y)
    types: ?x, ?y - bowl
     prec: ¬(?x = ?y) ∧ ¬holding() ∧ (number(?x) + number(y) < 3)
   effect: goods(?y) := goods(?y) + goods(?x),
           bads(?y) := bads(?y) + bads(?x),
           goods(?x) := 0, bads(?x) := 0

clean-bowl(?x)
    types: ?x - bowl
     prec: ¬holding()
   effect: goods(?x) := 0, bads(?x) := 0

sense-bowl(?x)
    types: ?x - bowl
     prec: ¬holding()
      obs: (bads(?x) = 0)

init: goods(small) := 0, bads(small) := 0,
      goods(large) := 0, bads(large) := 0,
      ¬holding(), ¬good-egg()
goal: (goods(small) = 3) ∨ (goods(large) = 3)
```

Figure 4.3: Description of the omelette problem with 3 eggs.

are not allowed to change denotation.[5]

4.4 Semantics

The semantics of the language is given in terms of induced trajectories within a formal transition system. This approach is chosen instead of a semantics based in FOL for several reasons, the most important being simplicity. A famous problem when trying to define a semantics based on FOL is the so-called *frame problem* in which the set of fluents (terms and relations) that do not change denotation need to be explicitly qualified in the logical representation of the transition functions [142, 182, 168]. With a semantics based on transition systems, the frame problem does not appear and the task becomes simpler. One advantage of a FOL-based semantics however is that it allows us to use FOL proof theories for reasoning about the effects of actions. This however is not necessary since such reasoning can be performed directly within the transition models. The kind of semantics found in this section has been used by [92]; semantics not based in FOL for other representation languages can be found, for example, in [156] for probabilistic independence and [98, 22] for counterfactuals and irrelevance in causal models.

Only the most general case of stochastic actions and partial information is considered since the other cases are formalized in a similar way but with simpler mathematical models. The mathematical models used in this case are the Partially Observable Markov Decision Processes (POMDPs) which are described in Ch. 3. A POMDP is fully characterized by the set of states, set of actions, transition probability matrices, observations probability matrices, cost functions and a distribution over the possible initial states. Thus, the task in this section is to define all these ingredients from a problem description as the one given in Fig. 4.3.

The road map is as follows. First, the basic elements of the description are used to define the set of constant, relational and functional symbols. Then, the set of operators and their effects are defined. Finally, the state space of the problem is defined as well as the action costs and the observation probabilities.

As will be seen, the method by which the problem description is formalized sheds light on how an implementation for a GPS based on this language can be designed. In fact, the GPT planning system is closely related to the method given below.

4.4.1 Actions and States

The formalization starts with the most basic elements supplied in the description: the types, objects, relational and functional symbols, and action symbols that define the sets $\mathcal{C}$, $\mathcal{K}$, $\mathcal{R}$, $\mathcal{F}$, and $\mathcal{A}$ respectively. In the omelette problem, these sets are:

$$\begin{aligned}
\mathcal{C} &= \{\,\texttt{bowl}, \texttt{integer}\,\},\\
\mathcal{K} &= \{\,\texttt{small}, \texttt{large}, 0, 1, 2, 3\,\},\\
\mathcal{R} &= \{\,\texttt{good-egg}, \texttt{holding}\,\},\\
\mathcal{F} &= \{\,\texttt{goods}, \texttt{bads}, \texttt{number}\,\},\\
\mathcal{A} &= \{\,\texttt{grab-egg}, \texttt{break-egg}, \texttt{pour-bowl}, \texttt{clean-bowl}, \texttt{sense-bowl}\,\}
\end{aligned}$$

plus the predefined relational and functional symbols '$<$' and '$+$'. Using the type information and the objects, the set of all operators A is defined by grounding the schemata with the corresponding objects.[6] Note that after grounding, the actions no longer contain quantified abbreviations like `exists` since all of them are substituted by their equivalent 'long' versions.

Given a state description S of the form (4.2), the result $S' = \text{result}(S, \alpha)$ of applying a single effect α in S is

$$S' \overset{\text{def}}{=} \{V'_f : f \in \mathcal{F}\} \cup \{V'_r : r \in \mathcal{R}\}$$

[5] Predefined functional and relational symbols behave like rigid designators in modal logic [128].

[6] This process substitutes variable by objects without any restriction of equality.

where $V'_f = V_f$ and $V'_r = V_r$ for all f and r except:

$$\begin{aligned} V'_r &= 1_{\mathcal{K}\setminus\{c\}} \cdot V_r \ + \ 1_{\{c\}} \cdot \text{true} && \text{if } \alpha = r(t), \\ V'_r &= 1_{\mathcal{K}\setminus\{c\}} \cdot V_r \ + \ 1_{\{c\}} \cdot \text{false} && \text{if } \alpha = \neg r(t), \\ V'_f &= 1_{\mathcal{K}\setminus\{c\}} \cdot V_f \ + \ 1_{\{c\}} \cdot c' && \text{if } \alpha = f(t) := t' \end{aligned}$$

where c, c' are the denotations of the terms t, t' respectively. The notation 1_A refers to the indicator function of the set A, and the arithmetic operations between functions refer to pointwise operators. Note the abuse of notation by treating tables V as functions and using arithmetic operations between them. In any case, the meaning should be clear and this abuse should cause no trouble. From now on, $s \sim S$ will denote that S is the description of state s. Additionally, the semantics is only given for unary symbols since the generalization to higher arity symbols is direct.

The result of applying a sequence $(\alpha_1, \ldots, \alpha_m)$ of simple effects is inductively defined as result$(S, (\alpha_1, \ldots, \alpha_m)) \overset{\text{def}}{=}$ result(result$(S, \alpha_1), (\alpha_2, \ldots, \alpha_m))$.

The evaluation of a formula in a state description is defined by means of a finite *computation tree* (CT) whose internal nodes correspond to boolean operations and leaves to boolean values. The definition of the CT for formulas not involving recursive predicates is straightforward. The case of recursive predicates is handled by constructing the *finite* CT that results by pruning the infinite branches with *neutral* values. A neutral value for the root node is defined and then propagated down the tree: the neutral value for 'and' nodes is true, for 'or' nodes is false, and *undefined* for negations. Thus, recursion through negation is not permitted.

Whenever a repeated recursive call (with respect to the upstream path to the root) is found, the node is pruned by replacing it with the neutral value. For example, Fig. 4.4 shows CTs for the `above(A,B)` predicate in the case of three blocks. The first tree is the initial CT with the recursive calls and the neutral values, while the second tree is the final (reduced) CT in which all infinite branches have been pruned. This reduced tree is further simplified to $(\texttt{pos(A) = B}) \vee ((\texttt{pos(A) = C}) \wedge (\texttt{pos(C) = B}))$ since $\texttt{pos}(x) = x$ is always false.

Given the restriction on the interpretations of formulas, the permitted recursive predicates can be translated into Logic Programs without functional symbols (also known as DATALOG) [135], and the CT semantics from above corresponds to a simple case of the stratified programs semantics of [4]. Yet, see [196] for an interesting discussion about the role of more general recursive predicates and their expressive power in planning formalisms.

Having defined the denotation of formulas in states, the result of a conditional effect '`when` φ `do` $\alpha_1, \ldots, \alpha_m$ `end-do`,' where $(\alpha_1, \ldots \alpha_m)$ is a sequence of *simple effects*, over a state description S is defined as the description

$$S' \overset{\text{def}}{=} \begin{cases} \text{result}(S, (\alpha_1, \ldots, \alpha_m)) & \text{if } S \text{ satisfies } \varphi, \\ S & \text{otherwise.} \end{cases}$$

Finally, the effect of a sequence $(\beta_1, \ldots, \beta_n)$ where each β_i is either a simple or conditional effect is defined inductively.

After the application of an action's effects, all ramification effects must be applied in order to obtain the resulting state.

The semantics for the effects of an applicable action a on state s, described by S and α respectively, is defined as $f(s, a) \overset{\text{def}}{=} s'$ with $s' \sim$ result$(S, (\alpha, \beta_1, \ldots, \beta_n))$ where the β_i are the effects associated with the ramification rules (following their static order). Thus, from now on, it will be assumed that the effects of an action a are 'extended' with the ramification effects.

For a non-deterministic effect $(\alpha_1, \ldots, \alpha_m)$ where each effect α_i is deterministic, the non-deterministic transition function is defined as $F(s, a) \overset{\text{def}}{=} \{s_1, \ldots, s_m\}$ where $s_i \sim$ result(S, α_i) and $s_i \models \varphi$ where φ stands for the conjunction of formulae associated with the logical invariants; see Appendix B for the syntax of logical invariants.

Finally, for a stochastic action $(\alpha_1, \ldots, \alpha_m)$ with probabilities $(p_1, \ldots, p_m)$ the transition probabilities are given by

$$p(s, a, s') \overset{\text{def}}{=} \sum \{ \, p_i : s' \sim \text{result}(S, \alpha_i), \ 1 \leq i \leq m \, \} \, .$$

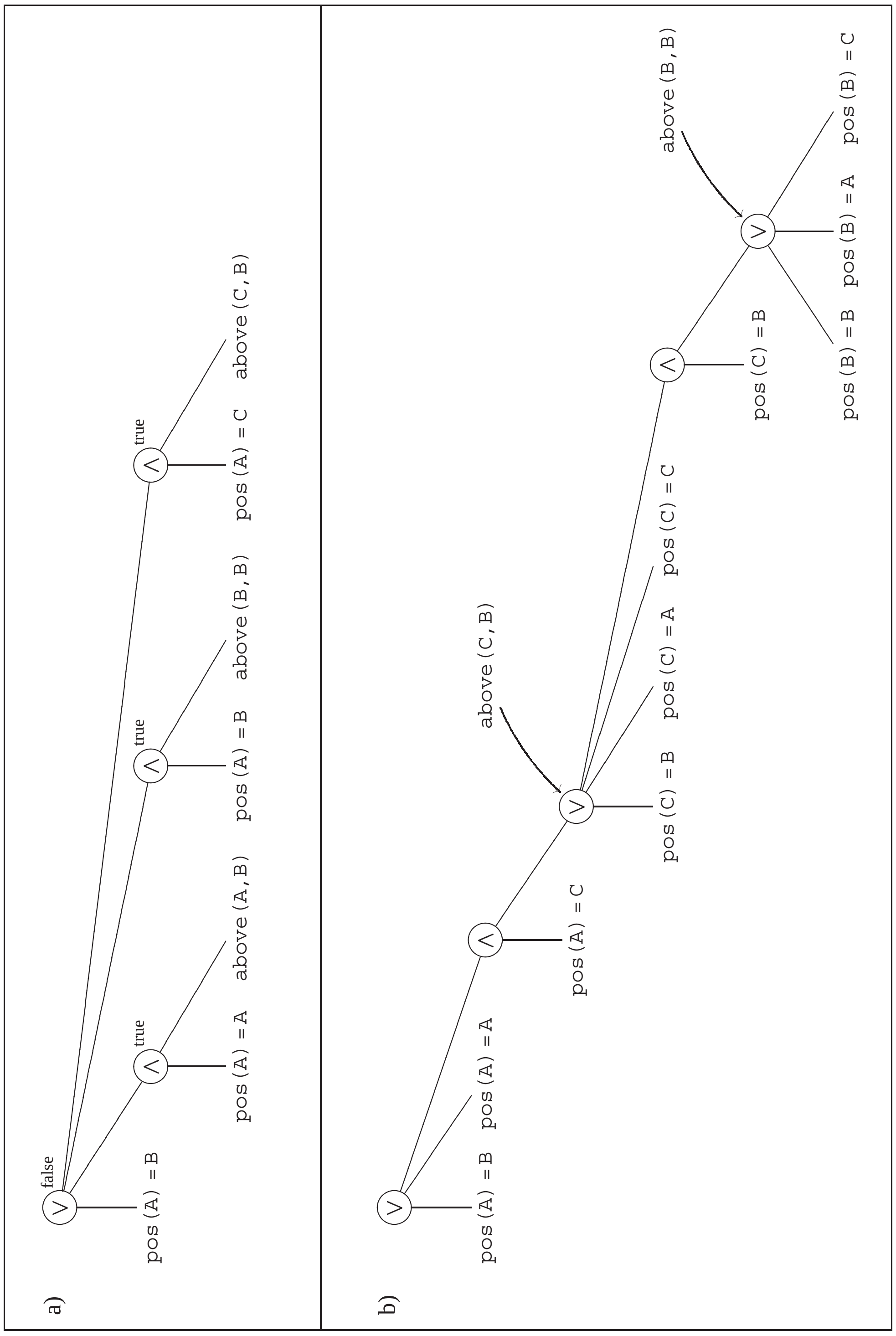


Figure 4.4: Computation trees for `above(A,B)`: (a) initial computation tree with neutral values labeling the nodes, (b) full computation tree after first reduction.

4.4.2 State Space

The state space of a problem is defined as the minimum set of states *stable* under application of actions, and containing the initial states. Thus, to specify the state space is enough to specify the set of initial states.

Each initial effect is either a deterministic effect, a set effect or a prune construct. The set of initial states is defined inductively as follows. Initially, there is a single state description given by $B_0 \overset{\text{def}}{=} \{S\}$ where $S = \{V_f : f \in \mathcal{F}\} \cup \{V_r : r \in \mathcal{R}\}$ is determined *statically* as $V_r \equiv false$ and $V_f \equiv c$ for c equal to the *first* constant symbol with the same type of f.

After having defined B_i, the set of descriptions B_{i+1} is given by using the $(i+1)$-th initial effect α_{i+1} as follows:

$$\begin{aligned} B_{i+1} &\overset{\text{def}}{=} \{\, \text{result}(S, \alpha_{i+1}) : S \in B_i \,\} && \text{if } \alpha_{i+1} \text{ is a deterministic effect,} \\ B_{i+1} &\overset{\text{def}}{=} \cup_{j=1}^{m} \cup_{S \in B_i} \{\, \text{result}(S, t := t_j) \,\} && \text{if } \alpha_{i+1} = \text{`}t := \{\, t_1, \ldots, t_m \,\}\text{'}, \\ B_{i+1} &\overset{\text{def}}{=} \{\, S \in B_i : S \text{ satisfies } \varphi \,\} && \text{if } \alpha_{i+1} = \text{`}\texttt{prune}\ \varphi\text{'}. \end{aligned}$$

Thus, the set of initial states is well defined and the state space can be computed by closing it with respect to the application of actions.

4.4.3 Action Costs and Observations

The definition of costs is done by direct induction on the structure of the cost descriptions. Let $G(S, a)$ denote the function that maps state descriptions S and cost formulas α to non-negative integers as follows:

$$\begin{aligned} G(S, \alpha) &\overset{\text{def}}{=} n && \text{if } \alpha \text{ is the integer } n, \\ G(S, \alpha) &\overset{\text{def}}{=} G(S, \alpha_1) + G(S, \alpha_2) && \text{if } \alpha = \text{`}\alpha_1 + \alpha_2\text{'}, \\ G(S, \alpha) &\overset{\text{def}}{=} \begin{cases} G(S, \alpha_1) & \text{if } S \text{ satisfies } \varphi, \\ G(S, \alpha_2) & \text{otherwise} \end{cases} && \text{if } \alpha = \text{`}\texttt{if}\ \varphi\ \alpha_1\ \alpha_2\ \texttt{fi}\text{'}. \end{aligned}$$

The action costs are then defined as $g(s, a) \overset{\text{def}}{=} G(S, \alpha)$ where $s \sim S$ and α is the cost formula for action a.

Finally, the set of observations are defined similarly so the details are left to the reader. If $O(s, a)$ is the set of possible observations after applying the action a in state s, the observation probabilities are defined as

$$q(s, a, o) \overset{\text{def}}{=} \sum \{\, p_i : o \sim O_i \,\}$$

where O_i is an observation description with probability p_i, and $o \sim O_i$ means that O_i is the description of the observation o.

4.5 Summary and Notes

This chapter presents the high-level description language used to express different planning tasks. The language is an instance of an action language based on the functional version of STRIPS given by [90], and is the base of the language within the GPT system [26].

The extension of STRIPS described in Section 4.2 corresponds to Pednault's ADL language [161] which is widely used in the classical planning setting.

The syntax of the language is an extension of McDermott's Planning Domain Definition Language or PDDL [145] for domains involving uncertainty and partial information. PDDL was the language used in the first planning competition that was held at the AIPS-98 conference; the results of the competition can be found in [144]. Since then, PDDL has become the de-facto standard description language for classical planning and has been used in the subsequent planning competitions. This is the main reason behind choosing a syntax so close to the PDDL language.

Recently, Rintanen [171] offered a study of the expressive power of the different formalisms used to describe general planning tasks. The description language presented here falls into his category of languages in Unary Non-determinism Normal form.

Other action languages are the A language of [93], the $\mathcal{AR}$ language used by [14], the probabilistic STRIPS language of [129], Reiter's and Sandewall's languages based on situation calculus [169, 179], and other formalisms based on dynamical Bayesian nets, decision trees, and factored models; e.g. [38, 133, 36].

Chapter 5

Algorithms for DDPs

This chapter presents different heuristic search algorithms like A*, IDA* and LRTA* for solving deterministic models. These algorithms provide the foundations for the algorithms developed in the next two chapters.

An optimal solution to a Deterministic Decision Process is a sequence of actions or steps that achieves a goal state from the initial state with minimum cost. Thus, a solution corresponds to a *minimum-cost* path in the *state graph* $G = (V, E, g)$ where V is the vertex (node) set, E is the edge set, and $g : E \to \mathbb{R}$ is a labeling of the edges with costs. The task of finding a minimum-cost path in graphs is a standard one in computer science and AI. This chapter presents the most important and relevant algorithms for this task from the perspective of heuristic search.

The algorithms can be naturally divided into uninformed or *blind search* algorithms, and informed or *heuristic search* algorithms. Furthermore, the search graph can be stored either *explicit* or *implicitly*. By an implicit graph, we mean a graph that is represented by a function that maps nodes into the set of its adjacent nodes.

In the case of constant costs, the most important blind search algorithm is Breadth-First Search or BFS, which finds a shortest path in $O(V + E)$ time [A1]; here, an explicit input graph is assumed so the space complexity is $\Theta(V + E)$. For general non-negative costs, the appropriate algorithm is the single-source shortest-path algorithm of Dijkstra's [66]. Dijkstra's algorithm computes all shortest paths from node s to every other node. The algorithm uses two pieces of information associated with each vertex v: an estimate of the distance from s to v denoted by $v.\text{DIST}$, and a pointer to a node u lying on a minimum-cost path $s \rightsquigarrow u \to v$ denoted by $v.\text{FATHER}$. The algorithm works with two sets Q and S of vertices and an operation that extracts a vertex $v \in Q$ with minimum distance $v.\text{DIST}$. Initially, all estimates are ∞ except $s.\text{DIST} = 0$, Q is the collection of all vertices and $S = \emptyset$. In each iteration of the algorithm, while Q is not empty, a node u with minimum (estimated) distance is moved from Q to S, and the distance of all neighbors of u is revised making them consistent with $u.\text{DIST}$. It can be shown, that each time a node u is inserted into the set S, the estimate $u.\text{DIST}$ is exact. Dijkstra's algorithm, shown in Alg. 1, [1] can be implemented in a number of ways, each yielding different running times. Thus, an implementation using a priority queue yields a running time $O(V^2 + E) = O(V^2)$, whereas for sparse graphs, i.e. $E = o(V^2/\log V)$, an implementation using binary min-heaps yields a running time of $O((V + E)\log V)$ [53]. In all cases, an explicit input graph is assumed. Other important algorithms are the Bellman-Ford algorithm for general cost functions, and the all-pairs shortest-paths algorithm of Floyd-Warshall [53].

The kind of problems that are encountered in AI have a huge number of states so algorithms that use explicit graphs are not practical. For example, by using algorithms tailored for search in implicit graphs, Korf was able to find optimal solutions for the Rubik's cube from random initial situations [123]. In his work, the size of the search space is 4.32×10^{19} and the solutions found have length between 16 and 18; the average *branching factor* is 13.34847. Clearly, standard graph algorithms cannot be applied to such big graphs.

An implicit search graph is defined in terms of a $\text{SUCCESSORS}(v)$ function, the cost function $g(u, v)$ and a goal testing function $\text{GOAL}(v)$. The complexity of the algorithms is measured in both time and space. Since space is usually the tightest constraint, the main interest is towards developing algorithms with low space consumption.

[1] All the algorithms in this dissertation are described using a notation close to that used in object-oriented programming languages.

```
DIJKSTRA($s$ : $state$)
begin
    // set $S$ stores nodes whose distance from $s$ have been
    // computed, and set $Q$ stores the remaining nodes

    // initialization
    $S := \emptyset$;
    $Q := V$;
    foreach vertex $v \in V$ do
        $v$.DIST $:= \infty$;
        $v$.FATHER $:= \perp$;
    $s$.DIST $:= 0$;

    // main loop: extract a node $u \in Q$ with minimum distance
    while $Q \neq \emptyset$ do
        $u := Q$.EXTRACTMINIMUM();
        $S := S \cup \{u\}$;

        // update distance for nodes adjacent to $u$
        foreach vertex $v$ adjacent to $u$ do
            if $v$.DIST $>$ $u$.DIST $+ g(u,v)$ then
                $v$.DIST $:=$ $u$.DIST $+ g(u,v)$;
                $v$.FATHER $:= u$;
end
```

Algorithm 1: Dijkstra's algorithm.

The version of Dijkstra's algorithm for implicit graphs is known as *uniform-cost search*. Uniform-cost search explores the nodes in the graph ordered by their costs until a goal node is found as shown in Alg. 2. As can be seen, the algorithm uses a priority queue $OPEN$ that stores the visited but not expanded nodes[2] ordered by their costs (distance from the start node), and a set $CLOSED$ that stores the expanded nodes. It this description, it is assumed that the priority queue $OPEN$ maintains at most one copy for each node in the graph. Thus, if a node is re-visited, the insert operation leaves $OPEN$ with a minimum cost version.

The optimality and correctness of uniform-cost search follows directly from the fact that the nodes are explored ordered by their costs. Its time and space complexities are proportional to the number of nodes with cost less than the optimal which is $O(b^c)$, where b is the branching factor and c is the minimum solution cost.

Uniform-cost search does not use any information other than the structure of the graph. More interesting algorithms are *informed search* algorithms that use heuristic functions to avoid exploring large portions of the graph.

5.1 Heuristic Search

Heuristic functions are value functions that are used to decide which nodes to *explore* first. Thus, a natural interpretation is to consider the heuristic function as information guiding the search. Since the information might not be accurate, algorithms should take the information given by the heuristic *cautiously*.

The simplest way to use a given heuristic h is to act *greedy* with respect to it. One possibility is that given a node v during the search, the greedy method always moves to a node $u \in$ SUCCESSORS(v) with minimum $h(u)$ value, i.e. nodes with less value are considered more desirable. The value $h(v)$ is then interpreted as an *estimate* of the cost

[2]A node is visited or generated if a data structure for it has been created in memory. A node is expanded or explored if all its successors nodes have been generated.

```
UNIFORMCOSTSEARCH($s$ : $state$)
begin
    // queue $OPEN$ stores visited but unexplored nodes
    // ordered by their costs, and $CLOSED$ stores explored nodes
    // initialization
    $CLOSED := \emptyset$;
    $OPEN :=$ THEEMPTYPRIORITYQUEUE;
    $OPEN$.PUSH($s, 0$);
    // main loop: extract a node $s \in OPEN$ of minimum cost
    $(s, cost) := OPEN$.GETMINIMUM();
    while $\neg s$.GOAL() do
        // explore $s$ and insert unexplored successors in $OPEN$
        $CLOSED := CLOSED \cup \{s\}$;
        for $s' \in s$.SUCCESSORS() $\setminus CLOSED$ do
            $new\_cost := cost + g(s, s')$;
            $OPEN$.INSERT($s', new\_cost$);
        // extract a node $s \in OPEN$ of minimum cost
        $(s, cost) := OPEN$.GETMINIMUM();
end
```

Algorithm 2: Uniform-cost search algorithm.

to reach the goal from node v. In this method, if the heuristic is an accurate estimate, the greedy algorithm finds a goal node quite fast and with constant memory consumption since no priority queue is involved, but if the heuristic is inaccurate the greedy algorithm may fail to find an optimal path, and even worst, may fail to find any path at all by getting trapped into a loop. To avoid the possibility of not finding a path from the start node to a goal node, the greedy method can be implemented with a priority queue $OPEN$ used to store the visited nodes and to avoid re-exploration of already expanded nodes, and in which the next node to explore is a node $v \in OPEN$ with minimum $h(v)$. This second greedy method is guaranteed to find a path from the initial node to a goal node, yet such path is not necessarily a minimum-cost path.

Despite these limitations of greedy methods, an interesting observation is worth mentioning [177]: while uniform cost search always *minimizes* the cost of the current path, greedy search always minimizes the estimated cost to the goal as given by h. Thus, the question is whether there is an algorithm which consider both the cost so far and the estimated remaining cost. The answer is yes, and the algorithm is known as A* [100, 153].

A* searches the graph systematically by minimizing the *node evaluation* function $f(v)$ given by

$$f(v) \overset{\text{def}}{=} g(v) + h(v)$$

where $g(v)$ is the cost of reaching node v from the initial state; i.e. if v was found in the path $s_0, s_1, \ldots, s_k = v$ then $g(v) = g(s_0, s_1) + \cdots + g(s_{k-1}, s_k)$. A* maintains a priority queue $OPEN$ with the visited nodes that have *not been explored* ordered by their f values, and a $CLOSED$ set that stores all explored nodes. Initially, $OPEN$ contains only the initial node and $CLOSED$ is empty. Then at each step, A* selects a node $v \in OPEN$ with minimum f value and checks whether v is a goal or not. If it is a goal, the algorithm finishes and a shortest path from the initial node to a goal node can be recovered. Otherwise, all unexplored successors of v are inserted into $OPEN$ and a new step begins. As in uniform-cost search, we assume that the $OPEN$ queue contains at most one copy of each node in the graph. Thus, if a node is re-visited only its less costly version is kept in $OPEN$. Alg. 3 contains a description of A*.

As all visited but not expanded nodes are always kept in the queue, it is easy to show that A* always terminates

```
A*($s : state$)
begin
    // priority queue $OPEN$ stores visited but unexplored nodes
    // ordered by their $f$ values, and $CLOSED$ stores explored nodes

    // initialization
    $CLOSED := \emptyset$;
    $OPEN :=$ THEEMPTYPRIORITYQUEUE;
    $OPEN$.PUSH($s, s$.H);

    // main loop: extract a node $s \in OPEN$ of minimum $f$ value
    $(s, f) := OPEN$.GETMINIMUM();
    while $\neg s$.GOAL() do
        // explore $s$ and insert unexplored successors in $OPEN$
        $CLOSED := CLOSED \cup \{s\}$;
        for $s' \in s$.SUCCESSORS() $\setminus CLOSED$ do
            $new\_f := f - s.\text{H} + g(s, s') + s'.\text{H}$;
            $OPEN$.INSERT($s', new\_f$);

        // extract a node $s \in OPEN$ of minimum $f$ value
        $(s, f) := OPEN$.GETMINIMUM();
end
```

Algorithm 3: A* algorithm.

at a goal state with no conditions on the heuristic whatsoever provided that all costs are positive and if such a path exists. Yet, for optimality one condition is needed. A non-negative heuristic function is said to be *admissible* if it always underestimates the minimum cost to the goal; i.e. $h(v) \leq J^*(v)$ where $J^*(v)$ denote the optimal cost from v to a goal. Then,

Theorem 19 (Hart, Nilsson and Raphael [100]) *Assume that all costs are positive and that there exists a path from the initial node to a goal node. If h is an admissible heuristic function,* A* *finds an optimal path from the initial node to a goal node.*

The proof of this theorem, which is not given here, is not difficult and can be found in any AI textbook, e.g. [177]. A very detailed analysis of A* and related search algorithms can be found in [155].

It is clear that for the identically zero heuristic $h \equiv 0$, A* becomes uniform-cost search. Therefore, its time and space complexities are (in the worst case) the same as the latter which is $O(b^c)$. With a more carefully analysis, it is not hard to show that A* in general explores all nodes v with value $f(v) < f(s_0)$ and that it may explore some (or all) nodes with $f(v) = f(s_0)$ where s_0 is the initial node [62]. Thus, its time and space complexity are $O(|\{v : f(v) \leq J^*(s_0)\}|)$.

Define an heuristic function as *monotonic* if the f value along any path from the initial node never *decreases*; in symbols

$$f(u) \;\leq\; \min\{\, f(v) : v \in \text{SUCCESSORS}(u)\,\}$$

for all u reachable from the initial node. Then, it can be shown that A* is *optimally efficient* given a monotonic heuristic [62]; that is, no other optimal algorithm working with the same heuristic explores fewer nodes than A*.

Despite its optimality, A* might still require exploring a large number of states if the heuristic function is not accurate enough; e.g. A* will run out of memory in matter of few minutes when attempting to solve Rubik's cube. Thus, other methods that tradeoff time with space are needed.

5.1.1 Linear-Space Algorithms

These algorithms perform multiple *bounded* Depth-First searches (DFS) looking for a goal node. At the beginning, the algorithms perform shallow searches and the bound is increased monotonically until a goal node is found. Since the interest is in optimality, the bound has to be increased conservatively as new information is obtained from previous searches. The best-known heuristic search algorithm in this class is Iterative Deepening A* (IDA*) [120]. Other linear space algorithms can be found in [176, 122, 101].

IDA* performs bounded depth-first searches each one involving only nodes with f value less than or equal to a given threshold. Initially, the threshold is equal to the heuristic value of the initial node. After each DFS finishes, the threshold is increased to the minimum f value of all visited but not explored nodes in the last DFS. Alg. 4 contains a recursive implementation of IDA* close to the one found in [177].

The correctness and optimality of IDA* follows from the way that the threshold value is increased after each DFS since two things are guaranteed: new nodes will be explored in the next DFS, and the first visited goal node lies on an optimal path. Thus, both conditions imply that eventually all nodes are visited and if a goal is found, it must be in an optimal path.

Since each DFS takes only space proportional to the maximum depth, the overall space consumption of IDA* is *linear* in the maximum search depth. As for the time complexity, consider a regular graph with branching factor b in which each bounded DFS with threshold t takes $O(b^t)$ time. Furthermore, assume that all costs are unitary and the worst-case heuristic $h \equiv 0$ so that the threshold starts at 0 and always increases by 1. Then, IDA* performs d depth-first searches with thresholds $t = 0, 1, \ldots, d$ for a total time complexity of

$$O(b^0 + b^1 + \cdots + b^d) \;=\; O(b^d).$$

This is the same complexity as for A*. A similar analysis carries for the general case of non-unitary costs. An analysis of the role of the heuristic function in IDA* appears in [125].

In summary, IDA* finds an optimal path in linear space and the same asymptotic time complexity of A*. For these reasons, IDA* is said to be *asymptotically optimal*.

However, when searching graphs and not trees, IDA* spends most of the time re-visiting duplicate nodes in the tree expansion of the graph; a problem that does not arise with A* since no node is ever expanded twice. This problem, which is caused by IDA*'s *lack of memory* of visited nodes, has been addressed by different authors with the common idea of using the available memory to store information about the visited nodes. See, for example, [176, 193, 149]; the origins of this idea can be traced back to [184]

Another way of using the available memory to improve IDA* is to store accurate pre-compiled heuristic functions known as *pattern databases* (see Ch. 8).

5.1.2 Learning Real-Time A*

Another interesting algorithm, that is generalized later for the case of non-deterministic models, is Korf's Learning Real-Time A* algorithm (LRTA*) [121]. Korf's motivation for this algorithm comes from a setting in which the agent interacts with the environment in real time, and the problem is to make decisions in order to reach a goal state.

This setting corresponds to a graph in which the edges (v, u) are labeled with decisions or actions a. Thus, $u \in \text{SUCCESSORS}(v)$ if and only if there exists an applicable action $a \in A(v)$ such that $u = f(v, a)$ where f is a deterministic transition function.

The LRTA* algorithm works by making at each step *greedy* decisions with respect to the heuristic function while *revising* the heuristic values so to avoid becoming trapped into a loop. That is, at each node v, the agent chooses an action a that minimizes the quantity

$$g(s, a) + f(v, a).\text{VALUE}$$

where $g(v, a)$ is the cost of applying action a at state v, $f(v, a)$ is the state that results after applying action a in state v, and $v.\text{VALUE}$ is the heuristic value for state v that initially is given by $h(v)$ but after applying action a in v is revised as

$$v.\text{VALUE} := \min_{a \in A(v)} g(v, a) + f(v, a).\text{VALUE}\,.$$

```
IDA*($s : state$)
begin
    // a global variable $goal$ that indicates whether a goal state
    // lying in a minimum-cost path has been found is used.
    // initialize threshold to $h(s)$ and goal $= false$
    $threshold := s$.H;
    $goal := false$;
    // main loop
    while $\neg goal$ do
        // perform a bounded DFS with given threshold
        $new\_threshold :=$ BOUNDEDDFS$(s, 0, threshold)$;
end

BOUNDEDDFS($s : state, cost : real, threshold : real$)
begin
    // check for bound using heuristic information
    if $cost + s$.H $> threshold$ then
        return $cost + s$.H;
    // check for goal condition
    else if $s$.GOAL() then
        $goal := true$;
        return $cost$;
    else
        // explore state and make recursive call
        $new\_threshold := \infty$;
        for $s' \in s$.SUCCESSORS() do
            $new\_cost := cost + g(s, s')$;
            $rv :=$ BOUNDEDDFS$(s', new\_cost, threshold)$;
            if $goal$ then return $rv$;
            $new\_threshold := \min\{rv, new\_threshold\}$;
        return $new\_threshold$;
end
```

Algorithm 4: IDA* algorithm.

```
LRTA*($s$ : $state$)
begin
    repeat TRIAL($s$);
end

TRIAL($s$ : $state$)
begin
    while $\neg s$.GOAL() do
        // pick best action and update hash
        $a := s$.GREEDYACTION();
        $s$.UPDATE();

        // simulate transition to next state
        $s := f(s, a)$;
end

$state$ :: GREEDYACTION()
begin
    return $\text{argmin}_{a \in A(s)}\ g(s, a) + f(s, a)$.VALUE;
end

$state$ :: UPDATE()
begin
    $a := s$.GREEDYACTION();
    $s$.VALUE $:= g(s, a) + f(s, a)$.VALUE;
end
```

Algorithm 5: LRTA* algorithm.

Each revision step can be understood as a *local* intervention of the heuristic function at a single state such that its value becomes *consistent* with the values of its successor states. Since the heuristic function is modified at a single state in each step, the (overall) modified function can be stored efficiently in a hash table H with a number of entries equal to the number of steps. Hence, the value of a state s is equal to $H(s)$ or $h(s)$ depending whether an entry for s exists in H. Alg. 5 shows a description of the LRTA* algorithm.[3]

As is shown by Korf, given any heuristic function, LRTA* is guaranteed to eventually reach the goal provided all costs are positive and that there are no dead ends in the underlying graph. Moreover, if the heuristic is admissible and LRTA* is applied multiple times to the same problem while preserving the hash table between the *episodes*, then LRTA* eventually behaves optimally. That is, LRTA* *learns* an optimal value function and then behaves optimally thereafter. This is the reason for the term 'learning' in the name of the algorithm.

The following two theorems show the convergence of LRTA* and bound the time of its convergence under the assumption of a monotonic heuristic function. The proof of the first theorem can be found in [121]. The proof of the second uses a result due to [119] which bounds the length of each LRTA* trial. Both theorems are proved in the next chapter in the more general context of non-deterministic systems.

Theorem 20 (Korf [121]) *Under the assumptions of a* DDP *model like D1–D6 (Ch. 3), that the goal is reachable from any state, and an admissible heuristic function h,* LRTA* *behaves optimally after a finite number of multiple trials.*

[3] In this description, the ties in the action selection step can be broken either randomly or in a systematic way.

Theorem 21 *Under the assumptions of a* DDP *model like D1–D6, that the goal is reachable from any state, and an admissible and monotonic heuristic function h, each* LRTA* *trial ends after at most $[\sum_{s \in S} J^*(s) - h(s) + J^*(s_0)]/\underline{g}$ steps, where $\underline{g}$ is the minimum positive cost and s_0 the initial state. Also, the total number of trials before convergence is at most the same quantity.*

Proof: Theorem 26 (in next chapter) shows that each LRTA* trial takes at most $[\sum_{s \in S} J^*(s) - h(s) + J^*(s_0)]/\underline{g}$ steps. In each trial, the monotonicity of the heuristic implies that if LRTA* has not yet converged, the value of a state is increased by at least $\underline{g}$. Hence, the total number of trials before convergence is bounded as claimed. □

5.2 Classical Planning as Heuristic Search

The problem of deterministic planning with uniform action costs, known as classical planning, can be naturally encoded into a DDP search problem. Often, we are only interested in finding a feasible plan (or sequence of actions) that maps the initial state to a goal state, yet recently some attention has been given to finding optimal plans [21, 116, 103, 107].

The main bottleneck for solving planning problems using explicit search in a state space is the lack of good heuristic functions. This problem caused a great deal of research looking for other formulations of the planning problem. Important formulations were given using SAT [115], constraint satisfaction [67], integer programming [199], etc. However, the discovery of *good* heuristics functions for classical planning and the success of standard algorithms using them have revived the idea of planning as heuristic search in a state space; see [143, 33, 25, 30, 108]. Indeed, in the recent planning competitions the planners based on heuristic search have shown impressive results [144, 6].

Conformant planning problems also correspond to deterministic decision processes as shown in Ch. 3. Thus, the algorithms in this chapter can be used to solve such problems given suitable heuristic functions. A recent formulation of probabilistic conformant planning using constraint satisfaction problems appears in [109].

The whole topic of heuristic functions for general planning tasks is treated in Ch. 8.

5.3 Summary and Notes

The standard algorithms for solving DDPs are presented in this chapter. The material is well known and can be found in standard texts of Computer Science and AI, e.g. [53, 177].

The LRTA* algorithm for on-line search is also presented. The LRTA* algorithm will serve as a basis for understanding the more sophisticated algorithms for MDPs given in the next chapter. Specifically, the labeled instances of the FIND-and-REVISE algorithms for MDP can be easily modified to obtain the corresponding labeled versions for deterministic decision processes. Thus, although such algorithms are not given in this chapter, we call the attention of the reader about these possibilities.

Chapter 6

Algorithms for MDPs

In this chapter, we present the standard value iteration algorithm for MDPs, the Real-Time Dynamic Programming algorithm (RTDP), the Loopy AO* algorithm (LAO*), and the novel algorithms based on heuristic search. All these algorithms are evaluated over a set of benchmark problems that have used by us and others research as a basic benchmark for evaluating MDP algorithms. As it will be seen, the new algorithms show better performance that the other algorithms.

We also present a series of results that proof the convergence and complexity properties of the new algorithms (Theorems 33–37). Additionally, this chapter contains a new proof for the convergence of RTDP as well as a new result that bound the expected termination time of each RTDP trial (Theorem 26).

In the case of Markov Decision Processes, a solution is not a sequence of actions but a policy or strategy that dictates what action to take for each state of the system. That is, a policy is a function that maps states to actions. As we saw in Ch. 3, it is sufficient to focus on policies whose actions does not depend on time, i.e. stationary policies, since it is known that there is an optimal policy that is stationary (Theorem 4).

The standard algorithm for MDPs, value iteration, finds an optimal policy that prescribes an action for all states of the system. However, in planning problems the interest is mainly in finding a policy that is closed with respect to the initial state s_0. Intuitively, a policy μ is closed with respect to s_0 if $\mu(s)$ is defined for all state s reachable from s_0 when the system is operated with μ.

Formally, if S_μ denotes the states for which the (possibly partial) policy μ is defined, and $\mu(A)$ is the set of possible successor states after applying μ in the set of states A, i.e.

$$\mu(A) \overset{\text{def}}{=} \{\, s' \in S : \exists(s \in A)(p(s,\mu(s),s') > 0)\,\},$$

then μ is said to be closed whenever $\mu(S_\mu) \subseteq S_\mu$, and said to be closed with respect to a state s if it is closed and $s \in S_\mu$. When $S_\mu = S$, then the policy μ is said to be *complete*.

The main motivation for focusing on partial policies instead of complete ones is that often complete optimal policies take too much space while closed optimal policies are very small. This phenomenon is due to the fact that the state space is exponential in the number of state variables yet only a fraction of the space is relevant when the sole interest is to find plan that takes the initial state to a goal state.

As an example, consider the (deterministic) problem of the Rubik's cube. Then, a optimal complete policy is one that tells what action to make in each possible configuration of the cube, while a minimal partial optimal policy with respect to the initial state is just a sequence of actions that take the initial state to the goal state. That is, if $s_0, \ldots, s_n$ is an optimal path from s_0 to the goal state for the Rubik's cube, then a partial policy that only is defined over such states is an optimal policy closed with respect to the initial state s_0.

This is analogous to the solutions provided by algorithms like Bellman-Ford versus A* in DDPs, as the former computes all shortest paths from the initial state while the latter computes only a shortest path from the initial state to a goal state.

As we will see, the value iteration algorithm finds complete optimal policies while the algorithms based on heuristic search find partial optimal policies that are closed with respect to the initial state.

All algorithms are given in their stochastic versions since the non-deterministic versions can be obtained by direct modifications.

6.1 Value Iteration

The value iteration algorithm is an direct consequence of Theorem 4 in Ch. 3. Remember that if J^* is the optimal value function for the MDP, then the greedy stationary policy

$$\mu^*(s) \overset{\text{def}}{=} \operatorname*{argmin}_{a \in A(s)} \left(g(s,a) + \alpha \sum_{s' \in S} p(s,a,s') J^*(s') \right)$$

is an optimal policy. Thus, for computing an optimal policy it is sufficient to compute the optimal value function that can be found using Bellman's equations as follows. Given any initial value function J_0, a new estimate is obtained by setting

$$J_{k+1}(s) \overset{\text{def}}{=} \min_{a \in A(s)} \left(g(s,a) + \alpha \sum_{s' \in S} p(s,a,s') J_k(s') \right)$$

for all $k > 0$. As it was shown in Theorem 4, $J_k \to J^*$ over all states. This is the standard value iteration algorithm for MDPs. (For stochastic shortest-path problems, the additional condition of $J_0(s) = 0$ for all goals $s \in S_G$ is required.) It can be easily verified that J_k is just the k-fold composition of T over J_0, i.e. $J_k = T^k J_0$. The operation of obtaining J_{k+1} from J_k is known as a (full) dynamic programming update or DP update.

The discounted case $\alpha < 1$ is of mathematical interest since it is easy to show that value iteration converges geometrically with rate α. Indeed, if T is the associated Bellman operator, then

$$\|J^* - J_k\| = \|T^k J^* - T^k J_0\| \leq \alpha^k \|J^* - J_0\|$$

and the geometric convergence follows at once.

In this case, value iteration is stopped as soon as the *residual*, defined as $\|J_{k+1} - J_k\|$, becomes sufficiently small, since then the suboptimality of the resulting policy can be easily bounded. Indeed, if μ_k is the greedy policy with respect to J_k, i.e. $T_{\mu_k} J_k = T J_k = J_{k+1}$, then

$$\begin{aligned} \|J_{\mu_k} - TJ_k\| &= \|T_{\mu_k} J_{\mu_k} - T_{\mu_k} J_k\| \\ &\leq \alpha \|J_{\mu_k} - J_k\| \\ &\leq \alpha \|J_{\mu_k} - J^*\| + \alpha \|J^* - J_k\|. \end{aligned}$$

On the other hand,

$$\begin{aligned} \|J_{\mu_k} - J^*\| &\leq \|J^* - TJ_k\| + \|TJ_k - J_{\mu_k}\| \\ &\leq \alpha \|J^* - J_k\| + \alpha \|J_{\mu_k} - J^*\| + \alpha \|J^* - J_k\| \\ &= 2\alpha \|J^* - J_k\| + \alpha \|J_{\mu_k} - J^*\|, \end{aligned}$$

and

$$\begin{aligned} \|J^* - J_k\| &\leq \|J^* - TJ_k\| + \|TJ_k - J_k\| \\ &\leq \alpha \|J^* - J_k\| + \|J_{k+1} - J_k\|. \end{aligned}$$

The desired result follows by substitution and collection of similar terms:

$$\|J^* - J_{\mu_k}\| \leq \frac{2\alpha}{(1-\alpha)^2} \|J_{k+1} - J_k\|.$$

As the number of stationary policies is finite, then for a sufficiently small residual the associated greedy policy is optimal.

In the case of undiscounted problems, no similar closed-form bound is known for the suboptimality of the resulting policies, yet other bounds can be computed at some cost [16]. It is reasonable to expect that for a sufficiently small residual, the corresponding greedy policy should be optimal for SSPs. This claim although intuitive is not direct and needs a proof. The following theorem and its proof establishes a somewhat known fact about SSPs, yet it does not appear explicitly in standard textbooks [164, 16].

Theorem 22 *There exists an $\epsilon > 0$ such that if the residual $\|J_{k+1} - J_k\| < \epsilon$, then the greedy policy with respect to J_k is optimal.*

6.1.1 Proofs

The following stronger result will be shown instead of Theorem 22.

Theorem 23 *Let $\{J_k\}_k$ be a sequence of value functions and $\{\epsilon_k\}_k$ a sequence of non-negative reals such that $\epsilon_k \to 0$ and $\|TJ_k - J_k\| \leq \epsilon_k$. Then, there exists n so that μ_k is optimal for all $k \geq n$, where μ_k is a greedy policy with respect to J_k.*

Lemma 24 *Let $\{J_k\}_k$ and $\{\epsilon_k\}_k$ be sequences of value functions and non-negative reals such that $\epsilon_k \to 0$ and $\|TJ_k - J_k\| \leq \epsilon_k$. Then, $J_k \to J^*$ where J^* is the fixed point of T.*

Proof: First note that all J_k are uniformly bounded since otherwise $\sup_k \|TJ_k - J_k\|$ would also be unbounded. Thus, J_k is a bounded sequence, and by compactness [A7], *convergent* subsequences $\{J_{k_j}\}_j$ can be extracted. The result will follow if it can be shown that any such subsequence converges to J^*. Let J' be the limit of J_{k_j}. Then,

$$\|TJ_{k_j} - J'\| \;\leq\; \|TJ_{k_j} - J_{k_j}\| + \|J_{k_j} - J'\|$$

which implies $TJ_{k_j} \to J'$ as the right-hand side can be as small as desired. The continuity of T implies $TJ_{k_j} \to TJ'$ so $TJ' = J'$. The result then follows since J^* is the unique fixed point of T. □

Proof of Theorem 23: The Lemma implies $J_k \to J^*$. By the continuity of T, $TJ_k \to J^*$. Let μ_k be a greedy policy with respect to J_k. Since the number of stationary policies is finite, $\{\mu_k\}_k$ is an infinite sequence of repeated policies. The theorem would be proved, if it can be shown that any policy that appears infinitely often is optimal. Let μ be any such policy. That is, $\mu = \mu_{k_j}$ where $\{\mu_{k_j}\}_j$ is a subsequence from $\{\mu_k\}_k$. Then,

$$T_\mu J_{k_j} \;=\; T_{\mu_{k_j}} J_{k_j} \;=\; TJ_{k_j} \;\to\; J^*$$

as $j \to \infty$. But the continuity of T_μ implies $T_\mu J_{k_j} \to T_\mu J^*$. Hence, $T_\mu J^* = J^*$ and μ must be optimal. □

6.2 Basic Benchmark

To evaluate the performance and benefits of the different MDP algorithms, the racetrack domain of Barto et al. [9] has been chosen as the *basic benchmark*.

This domain involves a racetrack or grid world in which a car has to be driven from an initial position to a set of goal positions. The states of the system are tuples (x, y, dx, dy) that denote the position and speed of the car along the x, y dimensions. The set of applicable actions are accelerations represented as pairs $a = (ax, ay)$ where $ax, ay \in \{-1, 0, 1\}$ and ax (ay) denotes the acceleration along the x (y) dimension. (A negative acceleration corresponds to applying the brakes.) Uncertainty in this domain comes from assuming that the road is 'slippery' which results in that the car may fail to respond to the actions with certain probability. More precisely, we will assume that an action $a = (ax, ay)$ has its intended effect

$$s' \;=\; (x + dx, y + dy, dx + ax, dy + ay)$$

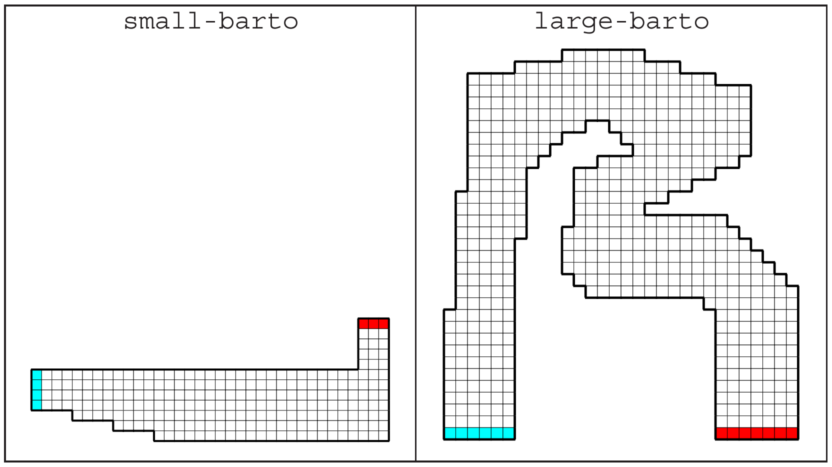


Figure 6.1: Racetracks small-barto and large-barto from [9]. The initial states are the light-shaded squares on the left of each figure, while the goal states are the dark-shaded squares on the right.

over state $s = (x, y, dx, dy)$ with probability $1 - p$, while with probability p the effects of the action correspond to that of the action $a_0 = (0, 0)$; i.e.,

$$s' = (x + dx, y + dy, dx, dy).$$

In order to reach a goal position, it is only necessary to cross over that position no matter the velocity of the car. Finally, it is assumed that whenever the car crashes against a wall, its velocity is reset to zero and its position is left intact (this is different from the formulation found in [9] where for some reason the position of the car is moved to the start position).

Examples of two racetracks are shown in Fig. 6.1. These two racetracks are the ones originally given in [9]. We will use these two racetracks plus others to evaluate the algorithms. Appendix Appendix C contains the racetracks for all instances.

The reasons for choosing this domain as a baseline for benchmark is that it offers the possibility of building a broad range of instances from easy to hard ones. We also perform experiments with values of $p = 0.1$ and $p = 0.2$, yet as it will be seen, the exact value of p does not affect the overall ranking of the algorithms.

6.3 Real-Time Dynamic Programming

The RTDP algorithm is the stochastic generalization of Korf's Learning Real-Time A* for deterministic heuristic search (Ch. 5). RTDP is a randomized learning algorithm for SSPs that computes a closed optimal policy with respect to the initial state by performing successive walks, also called trials, over the state space. Each RTDP trial starts at the initial state and finishes at a goal state. At all times t, the RTDP algorithm maintains an approximation J_t of J^* that is used to *greedily select* an action a_t to apply in the current state s_t.[1] Initially, J_0 is implicitly stored as an heuristic function $h(\,\cdot\,)$, and every time a control a_t is selected in state s_t, a new approximation J_{t+1} is computed by

$$J_0(s) \overset{\text{def}}{=} h(s), \tag{6.1}$$

$$J_{t+1}(s) \overset{\text{def}}{=} \begin{cases} J_t(s) & \text{if } s \neq s_t, \\ g(s_t, a_t) + \sum_{s' \in S} p(s_t, a_t, s') J_t(s') & \text{if } s = s_t. \end{cases} \tag{6.2}$$

Since J_t differs from J_0 at most in t states, J_t can be stored efficiently into a hash table H. Initially, H is empty and the value $H(s)$ is given by the heuristic $h(s)$. Thereafter, every time an action a_t is selected an update of the form of

[1]As in LRTA*, ties can be broken either randomly or in a systematic way.

```
RTDP($s$ : $state$)
begin
    repeat RTDPTRIAL($s$);
end

RTDPTRIAL($s$ : $state$)
begin
    // main loop
    while $\neg s$.GOAL() do
        // pick best action and update hash
        $a := s$.GREEDYACTION();
        $s$.UPDATE();

        // stochastically simulate next state
        $s := s$.PICKNEXTSTATE($a$);
end
```

Algorithm 6: RTDP algorithm.

(6.2) is applied to H such that J_t can be recovered from H and h. Alg. 6 and Alg. 7 show a schematic description of the RTDP algorithm and some basic functions respectively.

It is known that under assumptions A1 (there exists a proper policy) and A2 (all costs are positive) from Ch. 3, the RTDP trials *eventually* transverse minimum-cost paths from the initial state to a goal state if the heuristic function is *admissible*, i.e. if $0 \leq h(s) \leq J^*(s)$ for all $s \in S$ [9, 18]. Formally,

Theorem 25 *Consider an SSP model given by S1–S7 and assumptions A0–A2 from Ch. 3. Assume further that an infinite number of RTDP trials are performed each one starting at the initial state and ending at a goal state, and that the hash-table is preserved between trials. If h is admissible, then almost surely [A21]: (i) the sequence of vectors J_t generated by RTDP converges to (some) J_∞, (ii) $J_\infty(s) = J^*(s)$ for all recurrent states s, and (iii) the initial state is a recurrent state.*

The J_∞ vector in the theorem is a random vector that corresponds to the final hash table denoted by H_∞. Both objects J_∞ and H_∞ are random variables in the formal sense and the relation between them is that $J_\infty(s)$ is equal to $H_\infty(s)$ (respectively $h(s)$) if $s \in H_\infty$ (respectively if not). Part (iii) of the Theorem asserts that the greedy policy with respect to J_∞ is an optimal closed policy with respect to the initial state. Moreover, by Theorem 22, when the residual *over* the recurrent states is sufficiently small, the corresponding greedy policy is an optimal closed policy with respect to the initial state.

The standard proof of Theorem 25 given in [9, 18] is based on a version of value iteration known as *asynchronous* VI. The standard version, also referred as *synchronous* VI, performs a *full* DP update when computing the $(k+1)$-st iterate from the k-th iterate. In the asynchronous version, each DP does not need to be over all states simultaneously. The only requirement for convergence is that the value for each state should be updated infinitely often.

Thus, part of the claim in the theorem is that each sample path of the RTDP algorithm corresponds to asynchronous value iteration over a *random* SSP problem made of the recurrent states. That is to say, each sample path corresponds to solving a random MDP.

Instead of giving the standard proof of Theorem 25, a new proof that is simpler and self-contained is given. Additionally, when the heuristic function is not only admissible but also monotonic, the expected length of each RTDP trial can be bounded as it is shown in the following theorem. This result is one the theoretical contributions of this dissertation.

Theorem 26 *Consider an* SSP *model given by S1–S7 and assumptions A0–A2. Assume further that $p(s,a,s)=0$ for all non-goal states s and actions $a \in A(s)$. Let X_k be the state of the* RTDP *algorithm at time step k when started from the initial state s_0 and using the heuristic function h. If the heuristic function is admissible and monotonic, i.e. $0 \le h \le T\,h \le J^*$, then the expected hitting time of a goal state is bounded above by*

$$\underline{g}^{-1}\left(\sum_{s \in S}[J^*(s)-h(s)] \;+\; J^*(s_0)\right).$$

where $\underline{g}$ is the minimum positive cost.

Remark: In the deterministic setting, i.e. when the transition probabilities are zero or one, the theorem asserts that each LRTA* trial is bounded above by $\underline{g}^{-1}[\sum_{s\in S} J^*(s)-h(s)+J^*(s_0)]$.

Corollary 27 *Under the same assumptions of the theorem, every* RTDP *trial terminates at a goal step in a finite number of steps with probability one.*

6.3.1 Proofs

Lemma 28 *Let J be such $J \le TJ$, $\hat{s} \in S$ and define*

$$J'(s) \overset{def}{=} [\![s=\hat{s}]\!](TJ)(s) + [\![s \ne \hat{s}]\!]J(s).$$

Then, $J' \le TJ'$. In other words, DP *updates preserve monotonicity.*

Proof:

$$\begin{aligned}
(TJ')(s) &= \min_{a\in A(s)}\left(g(s,a) \;+\; \sum_{s'\in S} p(s,a,s')J'(s')\right)\\
&= \min_{a\in A(s)}\left(g(s,a) \;+\; \sum_{s'\ne\hat{s}} p(s,a,s')J'(s') \;+\; p(s,a,\hat{s})J'(\hat{s})\right)\\
&= \min_{a\in A(s)}\left(g(s,a) \;+\; \sum_{s'\ne\hat{s}} p(s,a,s')J(s') \;+\; p(s,a,\hat{s})(TJ)(\hat{s})\right)\\
&\ge \min_{a\in A(s)}\left(g(s,a) \;+\; \sum_{s'\ne\hat{s}} p(s,a,s')J(s') \;+\; p(s,a,\hat{s})J(\hat{s})\right)\\
&= (TJ)(s)\\
&\ge J'(s)
\end{aligned}$$

both inequalities by the monotonicity of J. □

Proof of Theorem 25: First assume that J is such that $J \le TJ$, and denote with $s_0, s_1, \ldots$ all states visited by RTDP over all trials; i.e. if the first trial ends in goal state s_t, make s_{t+1} the initial state of the second trial, etc.

Denote with J_t the value function at step t defined by the hash table at that time and the heuristic function, and let R be the set of *recurrent* states. Note that there is no change in the behavior of the RTDP algorithm if the value $J_t(s)$, for each time t, is replaced by a higher value for $s \notin R$. Since $0 \le J_t \le J^*$ for all t, it will be assumed without loss of generality that $J_t(s) = J^*(s)$ for all non-recurrent states s.

By $0 \le J_t \le J^*$ and compactness [A7], any subsequence of value functions contains *convergent* subsubsequences.

Let J_{t_i} be a convergent subsequence with limit $\hat{J}$. The goal is to show that $\hat{J}$ satisfies Bellman's equations and hence it must be J^*.

To establish the goal, extract a further subsequence $\{J_{t_{i(j)}}\}_j$ with the property that all recurrent states are visited between the times $t_{i(j-1)}$ and $t_{i(j)}$. Observe that the assumption $J \leq TJ$ implies that

$$TJ_{t_{i(j-1)}}(s) \;\leq\; J_{t_{i(j)}}(s)$$

for all recurrent states and also for the non-recurrent ones (since in that case the value is the optimal one). (This inequality is a direct consequence of Lemma 28.) Therefore, taking limits and using the continuity of the operator

$$T\hat{J} \;\leq\; \hat{J}$$

which is possible only if $\hat{J} = J^*$. Since the subsequence J_{t_i} was arbitrary then J_t converges to J^* over the recurrent states. The proof would be finished by showing that the goal state (and hence the initial state) is a recurrent state.

By Theorem 22, there exists an $\epsilon > 0$ such that if $\|J^* - J\| \leq \epsilon$ then the greedy policy with respect to J is optimal. Therefore, by $J_t \to J^*$, there is a finite time whence all actions chosen by RTDP for the recurrent states are optimal. Since the optimal policy is proper (see proof of Theorem 6), then the goal must be reached with probability one.

To remove the assumption $J \leq TJ$, consider a ‘parallel’ sequence of vector $\tilde{J}_k$ defined as

$$\begin{aligned}\tilde{J}_0(s) &\overset{\text{def}}{=} 0, \\ \tilde{J}_{k+1}(s) &\overset{\text{def}}{=} [\![s = s_k]\!](T\tilde{J}_k)(s) \;+\; [\![s \neq s_k]\!]\tilde{J}_k(s)\end{aligned}$$

for $s \notin R$ and equal to $J^*(s)$ for $s \in R$; i.e. a sequence of value functions defined as J_k but starting at the identically zero value function. Using induction it is not hard to see $\tilde{J}_k \leq J_k$ for all $k \geq 0$. Therefore, since $\tilde{J}_k$ converges to J^* over the recurrent states $J^* \leq \hat{J} \leq J^*$ and the proof is complete. □

The proof of Theorem 26 is obtained by studying the following stochastic process that starts at state s_0 and vector J given by

1. At time 0, let $X_0 = s_0$ and $J_0 = J$.
2. At each time k, choose an action $A_k \in A(X_k)$ greedy with respect to J_k.
3. Jump to state $X_{k+1} = s$ with probability $p(X_k, A_k, s)$.
4. Update vector as $J_{k+1}(s) = [\![X_k = s]\!](TJ_k)(s) + [\![X_k \neq s]\!]J_k(s)$.

Denote with $\hat{P}_{s_0,J}$ the probability measure associated with the process and with $\hat{E}_{s_0,J}$ the expectation with respect to this measure. Note that the process X_k mimics a RTDP trial starting at s_0 with heuristic function J. Without loss of generality, a single goal state t is assumed. Then, Theorem 26 follows from the stronger

Theorem 29 *Assume a model like S1–S7 and A0–A2, and that $p(s,a,s) = 0$ for all non-goal states s and actions $a \in A(s)$. Let τ be the (random) arrival time to the goal state, i.e. $\tau \overset{def}{=} \inf\{k > 0 : X_k = t\}$. Then,*

$$(k+1)\hat{P}_{s_0,J}(X_k \neq t) \;\leq\; \hat{E}_{s_0,J}(\tau) \;\leq\; \underline{g}^{-1}\left(\sum_{s\in S}[J^*(s) - J(s)] + J^*(s_0)\right)$$

for all J such that $J \leq TJ$.

Lemma 30 *If J is such that $0 \leq J \leq TJ$, then $0 \leq J \leq T^kJ \leq J^*$ for all $k \geq 0$.*

Proof: By the supposition and the monotonicity of T, $J \leq T^kJ$ for all k. But the right-hand side tends to J^* as $k \to \infty$. □

Proof of Theorem 29: Since $\hat{P}_{s_0,J}(\tau = 0) = 0$, the first inequality follows from the monotonicity of the events $\{X_k \neq t\} \supseteq \{X_{k+1} \neq t\}$ as

$$(k+1)\hat{P}_{s_0,J}(X_k \neq t) \;\le\; \sum_{j=0}^{k} \hat{P}_{s_0,J}(X_j \neq t) \;=\; \sum_{j=0}^{k+1} \hat{P}_{s_0,J}(\tau > j) \;\to\; \hat{E}_{s_0,J}(\tau)$$

as $k \to \infty$. For the second inequality, consider the sequence $(W_k)_{k\ge 0}$ defined as

$$\begin{aligned} W_0 &\overset{\text{def}}{=} 0, \\ W_{k+1} &\overset{\text{def}}{=} \sum_{s\in S}\big[J_{k+1}(s) - J(s)\big] - J_k(X_{k+1}) + J(X_0)\,. \end{aligned}$$

Then,

$$\begin{aligned} W_{k+1} &= \sum_{s\in S}\big[J_{k+1}(s) - J(s)\big] - J_k(X_{k+1}) + J(X_0) \\ &= \sum_{s\neq X_k}\big[J_{k+1}(s) - J(s)\big] + J_{k+1}(X_k) - J(X_k) - J_k(X_{k+1}) + J(X_0) \\ &= \sum_{s\neq X_k}\big[J_{k}(s) - J(s)\big] + J_{k+1}(X_k) - J(X_k) - J_k(X_{k+1}) + J(X_0) \\ &= \sum_{s\in S}\big[J_{k}(s) - J(s)\big] + J_{k+1}(X_k) - J_k(X_k) - J_k(X_{k+1}) + J(X_0) \\ &= \sum_{s\in S}\big[J_{k}(s) - J(s)\big] - J_{k-1}(X_k) + J(X_0) + J_{k+1}(X_k) - J_k(X_{k+1}) \\ &= W_k + J_{k+1}(X_k) - J_k(X_{k+1}) \end{aligned}$$

using the facts $J_{k+1}(s) = J_k(s)$ for $s \neq X_k$, and $J_k(X_k) = J_{k-1}(X_k)$ since $X_k \neq X_{k-1}$ by the assumptions. Let $\mathcal{F}_k = \{X_0, A_0, \ldots, X_k, A_k\}$ be the 'random history' of the process until time k (also known as the filtration). Taking expectations with respect to $\mathcal{F}_k$,

$$\begin{aligned} &\hat{E}_{s_0,J}\big[W_{k+1}\big|\mathcal{F}_k\big] \\ &= W_k + \hat{E}_{s_0,J}\big[J_{k+1}(X_k)\big|\mathcal{F}_k\big] - \hat{E}_{s_0,J}\big[J_k(X_{k+1})\big|\mathcal{F}_k\big] \\ &= W_k + \hat{E}_{s_0,J}\left[g(X_k,A_k) + \sum_{s\in S} p(X_k,A_k,s)J_k(s)\,\middle|\,\mathcal{F}_k\right] - \hat{E}_{s_0,J}\big[J_k(X_{k+1})\big|\mathcal{F}_k\big] \\ &= W_k + g(X_k,A_k) + \sum_{s\in S} p(X_k,A_k,s)J_k(s) - \hat{E}_{s_0,J}\big[J_k(X_{k+1})\big|\mathcal{F}_k\big] \\ &= W_k + g(X_k,A_k) + \sum_{s\in S} p(X_k,A_k,s)J_k(s) - \sum_{s\in S} p(X_k,A_k,s)J_k(s) \\ &= W_k + g(X_k,A_k) \\ &= W_k + [\![X_k \neq t]\!]g(X_k,A_k) \end{aligned}$$

The reader with background in probability theory will recognize W_k is a non-negative submartingale. Now, integrate both sides to obtain

$$\hat{E}_{s_0,J}\big(W_{k+1}\big) \;\ge\; \underline{g}\sum_{j=0}^{k} \hat{E}_{s_0,J}\big([\![X_j \neq t]\!]\big) \;=\; \underline{g}\sum_{j=0}^{k+1} \hat{P}_{s_0,J}(\tau > j)\,.$$

The admissibility of J (Lemma 30) implies $J_k \le J^*$ for all $k \ge 0$. Therefore,

$$\sum_{s\in S}[J^*(s) - J(s)] + J^*(X_0) \;\ge\; \hat{E}_{s_0,J}(W_{k+1}) \;\ge\; \underline{g}\sum_{j=0}^{k+1} \hat{P}_{s_0,J}(\tau > j) \;\to\; \underline{g}\hat{E}_{s_0,J}(\tau)\,.$$

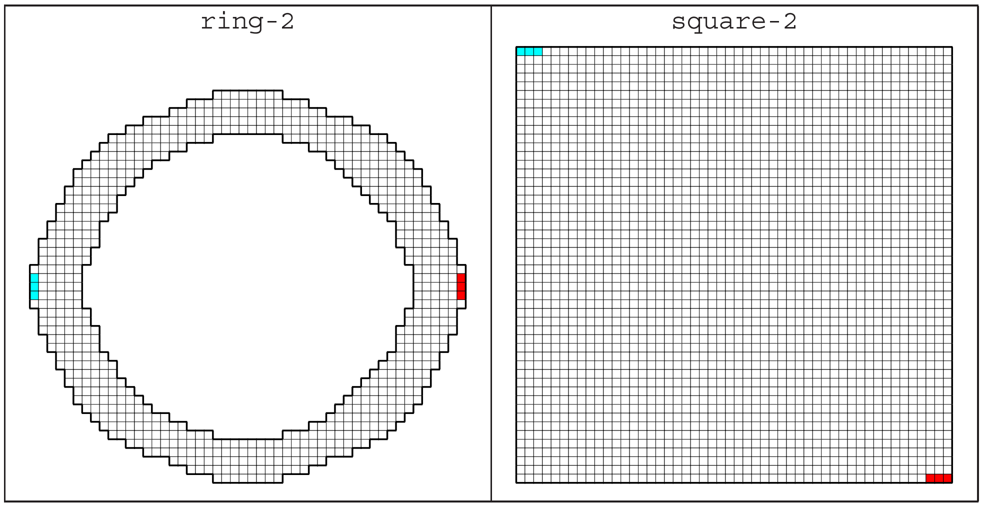


Figure 6.2: Racetracks `ring-2` and `square-2` The initial states are the light-shaded squares on the left of each figure, while the goal states are the dark-shaded squares on the right.

The first inequality follows by the definition of W_k, and the limit by using $\hat{E}_{s_0,J}(\tau) = \sum_{j\geq 0} \hat{P}_{s_0,J}(\tau > j)$ since $\hat{P}_{s_0,J}(\tau = 0) = 0$. $\square$

6.4 Convergence and Anytime Behavior of RTDP

From a practical point of view, RTDP has positive and negative features. The positive feature is that RTDP has a good *anytime behavior*: it can quickly produce a good policy and then smoothly improve it with time. The negative feature is that *convergence*, and hence termination, is slow. These two features are illustrated by comparing the anytime and convergence behavior of RTDP in relation to value iteration over a couple of experiments. The anytime behavior of these methods is displayed by plotting the average cost to the goal of the greedy policy μ_t with respect to the value function J_t as a function of time t. This cost changes in time as a result of the updates on the value function. The averages are taken by running 500 simulations after every 25 trials for RTDP, and 500 simulations after each full DP update for VI. The curves in Fig. 6.3 are further smoothed by a moving average over 50 points (RTDP) and 5 points (VI).

For the experiments two racetracks shapes were considered: one is a ring with 33068 states (`ring-2`), the other is a full square with 383950 states (`square-2`). These racetracks are shown Fig. 6.2. In both cases, the task is to drive the car from one extreme in the racetrack to its opposite extreme. Optimal policies for these problems turn out to have expected costs (number of steps) 14.96 and 10.48 respectively, and the percentages of relevant states (i.e. those reachable through an optimal policy) are 11.47% and 0.25% respectively.

Fig. 6.3 shows the quality of the policies found as a function of time for both value iteration and RTDP. In both cases $J_0 = h \equiv 0$ is used as the initial value function; i.e. no (informative) heuristic is used in RTDP. Still, the quality profile for RTDP is much better: it produces a policy that is practically as good as the one produced eventually by VI in almost half of the time. Moreover, by the time RTDP yields a policy with an average cost close to optimal, VI yields a policy with an average cost that is more than 10 times higher.

The problem with RTDP is *convergence*. Indeed, while in these examples RTDP produces near-optimal policies at times 3 and 40 seconds respectively, it converges, in the sense defined above, in more than 10 *minutes*. Value iteration, on the other hand, converges in 5.435 and 78.720 seconds respectively.

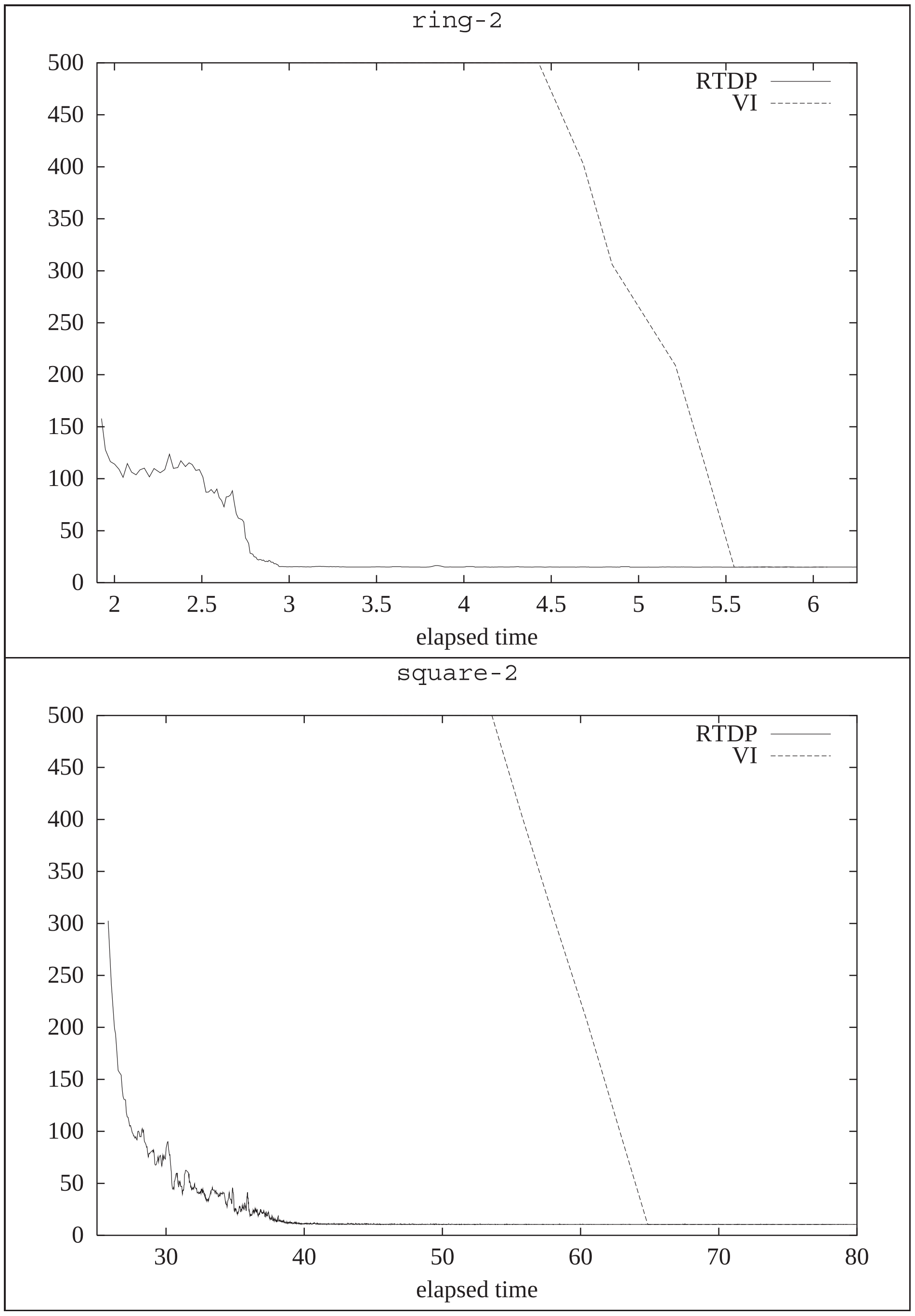


Figure 6.3: Quality profiles: Average cost to the goal vs. time for value iteration and RTDP over two racetracks.

6.5 Find-and-Revise Algorithms

The FIND-and-REVISE schema is a general asynchronous VI algorithm that exploits knowledge of the initial state and an admissible heuristic for computing optimal or near-optimal policies without having to evaluate the entire space. Let us say that a value function J is *ϵ-consistent* (inconsistent) over a state s when the residual over s is no greater (greater) than ϵ, and that J itself is ϵ-consistent when it is ϵ-consistent over all the states reachable from s_0 and the greedy policy μ_J (greedy with respect to J).[2] Then FIND-and-REVISE computes an ϵ-consistent value function by simply searching for inconsistent states in the greedy graph and updating their values until no such states are left. Alg. 8 shows a description of the FIND-and-REVISE algorithm.

The *greedy graph* G_J refers to the graph resulting from the execution of the greedy policy μ_J starting in the initial state s_0; i.e., s_0 is the single root node in G_J, and for every non-goal state s in G_J, its children are the states that may result from executing the action $\mu_J(s)$ in s.

The procedures FIND and REVISE are the two parameters of the FIND-and-REVISE procedure. For convergence, optimality, and complexity of FIND-and-REVISE, it is assumed that FIND searches the graph systematically, and REVISE of J at s updates J at s (and possibly at some other states), both operations taking $O(|S|)$ time.

The properties of the FIND-and-REVISE algorithms are established for a class of initial value functions (heuristics) that satisfy the properties $0 \leq J \leq J^*$ and $0 \leq J \leq TJ$ which are known respectively as the heuristic is *admissible* and *monotonic*. Admissible value functions were used before to establish the convergence of the RTDP algorithm. Monotonicity on the other hand is a stronger property that also appears in the case of deterministic models (see Ch. 5). The fact that monotonicity implies admissibility is a direct consequence of the monotonicity of the operator T. Indeed, since $TJ \leq TJ'$ for $J \leq J'$, then by induction $J \leq T^k J$ and the latter goes to J^*. DP updates not only preserve the admissibility of value functions but also the monotonicity property (Lemma 28).

The following results for FIND-and-REVISE are direct consequences of the assumed properties on the FIND and REVISE procedures, and the heuristic function.

Theorem 31 *For a model given by S1–S7 and assumptions A0–A2, and an initial value function (heuristic) h that is admissible and monotonic,* FIND*-and-*REVISE *yields an ϵ-consistent value function in a number of loop iterations no greater than $\epsilon^{-1} \sum_{s \in S} J^*(s) - h(s)$, where each iteration has time complexity $O(|S|)$.*

Theorem 32 *For a model given by S1–S7 and assumptions A0–A2, and an initial value function (heuristic) h that is admissible and monotonic, the value function computed by* FIND*-and-*REVISE *and the corresponding greedy policy approach the optimal value function and an optimal policy over all relevant states as $\epsilon \to 0$.*

6.5.1 LAO* and Improved LAO*

Hansen and Zilberstein's LAO* algorithm [99] is an algorithm for computing closed policies with respect to the initial state. Similar to the FIND-and-REVISE algorithm, LAO* computes an (ϵ-)consistent value function J over all states reachable from the initial state on the greedy graph G_J.

LAO* works by maintaining an explicit graph G over states which is extended until the graph has no more *tip* nodes and the residual falls below some given ϵ. The pseudo-code for LAO* is given in Alg. 9.

As it can be seen, the LAO* algorithm can be understood as an instance of the general FIND-and-REVISE scheme where the FIND operation is carried by the expansion of the graph in the line marked 2, while the REVISE operation is given by the value iteration algorithm over the states in the current explicit graph. However, although the complexity of the FIND operation is $O(|S|)$ time, the REVISE operation is often more costly. Hence, Theorem 31 does not hold for LAO*.

Indeed, a faithful implementation of LAO* results in an algorithm that is much worst than value iteration over the racetrack benchmark. For this reason, Hansen and Zilberstein proposed a variation of LAO*, called improved LAO* or ILAO*, which is similar to some instances of FIND-and-REVISE (see below). ILAO* is like LAO* but with a less expensive update operation in which the values of all nodes in the greedy graph are updated only once in a depth-first order. The pseudo-code for the ILAO* algorithm is given in Alg. 10.

[2]From now on, μ_J will denote the greedy policy with respect to a value function J without further comment.

6.5.2 Labeled Real-Time Dynamic Programming

The fast improvement phase in RTDP as well as its slow convergence phase are both consequences of an exploration strategy — greedy simulation — that privileges the states that occur in the most likely paths resulting from the greedy policy. These are the states that are most relevant given the current value function, and that's why updating them produces such a large impact. Yet states along less likely paths are also needed for convergence, but these states appear less often in the simulations. This suggests that a potential way for improving the convergence of RTDP is by adding some *noise* in the *simulation*.[3] Here, however, a different approach is taken that has interesting consequences both from a practical and a theoretical point of view.

Roughly, rather than forcing RTDP out of the most likely greedy states, a record of the states over which the value function has converged is kept to avoid re-visiting them during future searches. This is accomplished by means of a *labeling procedure* that is called CHECKSOLVED. Let us refer to the graph made up of the states that are reachable from a state s with the greedy policy (for the current value function J) as the *greedy graph* and let us refer to the set of states in such a graph as the *greedy envelope*. Then, when invoked in a state s, the procedure CHECKSOLVED searches the greedy graph rooted at s for a state s' with residual greater than ϵ. If no such state s' is found, and only then, the state s is labeled as SOLVED and CHECKSOLVED(s) returns true. Namely, CHECKSOLVED(s) returns true and labels s as SOLVED when the current value function has converged over s. In that case, CHECKSOLVED(s) also labels as SOLVED all states s' in the greedy envelope of s as, by definition, they must have converged too. On the other hand, if a state s' is found in the greedy envelope of s with residual greater than ϵ, then this and possibly other states are updated and CHECKSOLVED(s) returns false. These updates are crucial for accounting for some of the theoretical and practical differences between RTDP and the labeled version of RTDP that is called *Labeled* RTDP or LRTDP.

Note that due to the presence of cycles in the greedy graph it is not possible in general to have the type of recursive, bottom-up labeling procedures that are common in some AO* implementations [153], in which a state is labeled as solved when its successors states are solved. Such a labeling procedure would be sound but incomplete, namely, in the presence of cycles it may fail to label as solved states over which the value function has converged. Below, more will be said about labeling in the presence of cycles.

The code for CHECKSOLVED is shown in Alg. 11. The label of state s in the procedure is represented with a bit in the hash table denoted with s.SOLVED. Thus, a state s is labeled as solved iff s.SOLVED $=$ true. Note that the depth-first search stores all states visited in a $closed$ list to avoid loops and duplicate work. Thus the time and space complexity of CHECKSOLVED(s) is $O(|S|)$ in the worst case, while a tighter bound is given by $O(|S_J(s)|)$ where $S_J(s) \subseteq S$ refers to the greedy envelope of s for the value function J. This distinction is relevant as the greedy envelope may be much smaller than the complete state space in certain cases depending on the domain and the quality of the initial value (heuristic) function.

An additional detail of the CHECKSOLVED procedure shown in Alg. 11 is that for performance reasons, the search in the greedy graph is not stopped when a state s' with residual greater than ϵ is found; rather, all such states are found and collected for update in the $closed$ list. Still such states act as *fringe* states and the search does not proceed beneath them (i.e. their children in the greedy graph are not added to the $open$ list).

In presence of a monotonic value function J, DP updates have always a *non-decreasing* effect on J. As a result, the following key property can be easily established:

Theorem 33 *Assume that J is an admissible and monotonic value function. Then, a* CHECKSOLVED(s, ϵ) *call either labels a state as solved, or increases the value of some state by more than ϵ while decreasing the value of none.*

A consequence of this result is that CHECKSOLVED can solve by itself models of the form S1–S7. Indeed, if a model is said to be *solved* when a value function J converges over the initial state, then it is simple to prove that:

Theorem 34 *Provided that the initial value function is admissible and monotonic, and that the goal is reachable from every state, then the procedure:*

[3] This is different than adding noise in the action selection rule as done in reinforcement learning algorithms [190]. In reinforcement learning, noise is needed in the action selection rule to guarantee optimality while no such thing is needed in RTDP. This is because full DP updates as used in RTDP, unlike partial DP updates as used in Q or TD-learning, preserve admissibility.

```
    while $\neg s_0$.SOLVED do CHECKSOLVED$(s_0, \epsilon)$
```

solves the model S1–S7 under assumptions A0–A2 in a number of iterations no greater than $\epsilon^{-1} \sum_{s \in S} J^*(s) - h(s)$, *where each iteration has complexity* $O(|S|)$.

The solving procedure in Theorem 34 is an instantiation of the general FIND-and-REVISE schema that is novel and interesting although it is not sufficiently practical. The problem is that it does not attempt to label other states as $solved$ before labeling s_0. Yet, unless the greedy envelope is a strongly connected graph, s_0 will be the last state to converge and not the first. In particular, states that are close to the goal will normally converge faster than states that are farther.

This is the motivation for the alternative use of the CHECKSOLVED procedure in the *Labeled* RTDP *algorithm*. LRTDP trials are very much like RTDP trials except that trials terminate when a $solved$ stated is reached (initially only the goal states are solved) and they invoke then the CHECKSOLVED procedure in reverse order, from the last *unsolved* state visited in the trial back to s_0, until the procedure returns false on some state. Labeled RTDP terminates when the initial state s_0 is labeled as SOLVED. The code for LRTDP is shown in Alg. 12. The first property of LRTDP is inherited from RTDP:

Theorem 35 *Provided that the goal is reachable from every state and the initial value function is admissible, then* LRTDP *trials cannot be trapped into loops, and hence must terminate in a finite number of steps.*

The novelty in the optimality property of LRTDP is the bound on the total number of trials required for convergence:

Theorem 36 *Provided that the goal is reachable from every state and the initial value function is admissible and monotonic, then* LRTDP *solves the model S1–S7 in a number of trials bounded by* $\epsilon^{-1} \sum_{s \in S} J^*(s) - h(s)$.

The LRTDP algorithm can be thought as an instantiation of the general FIND-and-REVISE schema in which the FIND procedure makes a *randomized* search for a state with residual bigger than ϵ, and in which a labeling mechanism is maintained in order to avoid revisits. As it will be seen next, LRTDP improves the convergence time of RTDP dramatically while maintaining its good anytime behavior.

Proofs

Proof of Theorem 33: A state in the greedy graph rooted at state s has residual $> \epsilon$ if and only if $rv = false$ after the main loop in Alg. 11. In such case, the value for some state s' (and possibly others) is updated increasing its value in more than ϵ. Since the heuristic is monotonic no value is ever decreased. In $rv = true$ at the end of the main loop, the state s and all states in the greedy graph rooted at s are labeled as solved. □

Proof of Theorem 34: Each call of CHECKSOLVED either increases the value of some state or labels a state as solved. Thus, since the values are bounded above by J^*, a finite number of calls to CHECKSOLVED are sufficient to label s_0 as solved. The number of calls is clearly bounded by the quantity in the theorem. □

Empirical Evaluation

In this section, LRTDP is evaluated empirically in relation to RTDP and two other DP algorithms: value iteration, the standard dynamic programming algorithm, and LAO*. Value iteration is the baseline for performance; it is the standard algorithm that is very practical yet it computes complete policies and needs as much memory as states in the state space. The other three algorithms LAO*, RTDP and LRTDP compute only partial optimal policies and do not need to consider the entire space. The algorithms are compared along two dimensions: anytime and convergence behavior. The anytime behavior is evaluated by plotting the average cost of the policies found by the different methods as a function of time, as described in the section on convergence and anytime behavior. Likewise, convergence time is evaluated by displaying the time and memory (number of states) required.

problem	$\lvert S\rvert$	$J^*(s_0)$	% rel.	$h_{min}(s_0)$	time h_{min}
small-b	9,312	11.084923744201	13.96	10.00	$1.008 \pm .004$
large-b	23,880	17.147188186645	19.36	16.00	$3.208 \pm .166$
h-track	53,597	38.433315277099	17.22	36.00	$13.603 \pm .741$
y-1	81,775	13.764185905456	1.35	13.00	$12.170 \pm .393$
y-2	239,089	15.462683677673	0.86	14.00	$39.827 \pm .558$
square-1	42,071	7.508861064910	1.67	7.00	$4.657 \pm .241$
square-2	383,950	10.484800338745	0.25	10.00	$59.671 \pm .484$
ring-1	5,895	10.377809524536	10.65	10.00	$0.548 \pm .003$
ring-2	33,068	14.960873603820	11.47	14.00	$4.760 \pm .004$

Table 6.1: Information about the different racetrack instances with parameter $p = 0.1$: size, expected cost of the optimal policy from s_0, percentage of relevant states, heuristic for s_0 and time spent in seconds for computing h_{min}. All times are in seconds.

The experiments are performed with a *domain-independent* heuristic, called h_{min}, for initializing the value function and with the heuristic function $h = 0$. The former is obtained by solving a simple relaxation of Bellman's equation

$$J(s) \;=\; \min_{a \in A(s)} \left(g(s,a) + \sum_{s' \in S} p(s,a,s')\, J(s') \right)$$

to the simpler equation

$$J(s) \;=\; \min_{a \in A(s)} \big(g(s,a) + \min \{\, J(s') : p(s,a,s') > 0 \,\} \big). \tag{6.3}$$

That is, by replacing the expected value over the possible successor state by the minimum such value. This relaxation corresponds to a problem in which each uncertain outcome is replaced by the less costly outcome (i.e. the most favorable).

Clearly, the optimal value function of the relaxation (i.e. the solution to (6.3)) is a lower bound for the Bellman equation. The h_{min} heuristic is defined as the optimal value function associated to (6.3).

In the experiments, h_{min} is computed on need by running a labeled variant of the LRTA* algorithm (the deterministic variant of RTDP) until convergence, from the state s where the heuristic value $h(s)$ is needed. Once again, unlike standard DP methods, LRTA* does not require evaluating the entire state space in order to compute these values, and moreover, a separate hash table containing the values produced by these calls is kept in memory, as these values are reused among calls (this is like doing several single-source shortest path problems on demand, as opposed to a single but potentially more costly all-sources shortest-path problem when the set of states is very large). The total time taken for computing these heuristic values in the heuristic search procedures like LRTDP and LAO*, as well as the quality of these estimates, will be shown separately below. Briefly, this time is seen to be significant and often more significant than the time taken by LRTDP. Yet, it is also seen that the overall time remains competitive with respect to value iteration and to the same LRTDP algorithm with the $h = 0$ heuristic. It is not hard to show that this heuristic is admissible and monotonic (see Ch. 8).

Problems

The problems considered are all instances of the racetrack domain. They are the instances `small-b` and `large-b` from [9], `h-track` from [99],[4] and three other tracks, `ring`, `square`, and `y` (2 versions) with the corresponding shapes (shown in Appendix C). Table 6.1 contains relevant information about these instances: their size, the expected cost of the optimal policy from s_0, and the percentage of relevant states.[5] The last two columns show information

[4]Taken from the source code of LAO*.

[5]All these problems have multiple initial states so the value $J^*(s_0)$ (and $h_{min}(s_0)$) in the table is the average over the initial states.

algorithm	`small-b`	`large-b`	`h-track`	`y-1`	`y-2`
VI$(h=0)$	1.098	**3.942**	19.718	16.266	62.361
ILAO*$(h=0)$	2.524	12.683	45.068	58.487	187.054
LRTDP$(h=0)$	**0.875**	6.057	**15.277**	**10.083**	**34.984**
VI(h_{min})	1.032	3.478	11.764	15.064	55.864
ILAO*(h_{min})	0.575	2.882	14.631	0.538	1.124
LRTDP(h_{min})	**0.385**	**2.140**	**7.143**	**0.239**	**0.492**

algorithm	`square-1`	`square-2`	`ring-1`	`ring-2`
VI$(h=0)$	6.580	84.301	0.630	5.348
ILAO*$(h=0)$	12.875	238.236	1.106	11.019
LRTDP$(h=0)$	**3.332**	**50.573**	**0.466**	**3.777**
VI(h_{min})	5.824	79.876	0.561	6.651
ILAO*(h_{min})	0.302	1.148	0.372	3.682
LRTDP(h_{min})	**0.166**	**0.321**	**0.128**	**1.366**

Table 6.2: Convergence time in seconds for the value iteration, ILAO*, and LRTDP algorithms with the initial value functions $h = 0$ and h_{min} and $\epsilon = 10^{-3}$ for several racetrack instances with parameter $p = 0.1$. Times for RTDP not shown as they exceed the cutoff time for convergence (10 minutes). All times are in seconds, and the fastest ones are shown in bold.

about the heuristic h_{min}: the value for $h_{min}(s_0)$ and the total time in second for computing the heuristic values in the LAO*, LRTDP (and HDP from next section) algorithms for the different instances. Interestingly, these times are roughly equal for all algorithms across the different problem instances. Note that the heuristic provides a pretty tight lower bound yet it is somewhat expensive. Below, the experiments are run with both the heuristic h_{min} and the heuristic $h = 0$.

Algorithms

The algorithms used in the comparison are VI, RTDP, LRTDP, and the variant of LAO*, called Improved LAO* (ILAO*). The four algorithms have been implemented in C++. The results have been obtained on a Sun-Fire 280-R with 1Gb of RAM and a clock speed of 750Mhz.

Results

Curves in Fig. 6.4 display the evolution of the average cost to the goal as a function of time for the different algorithms, with averages computed as explained before, for two problems: `ring-2` and `square-2`. The curves correspond to $\epsilon = 10^{-3}$ and $h = 0$. RTDP shows the best profile, quickly producing policies with average costs near optimal, with LRTDP close behind, and VI and ILAO* further behind.

The top three rows of Table 6.2 shows the times needed for convergence (in seconds) for $h = 0$ and $\epsilon = 10^{-3}$ (results for other values of ϵ are similar). The times for RTDP are not reported as they exceed the cutoff time for convergence (10 minutes) in all instances. Note that for the heuristic $h = 0$, LRTDP converges faster than VI in 8 of the 9 problems. ILAO* takes longer yet also solves all problems in a reasonable time. The reason that LRTDP, like RTDP, can behave so well even in the absence of an informative heuristic function is because the focused search and updates quickly boost the values that appear most relevant so that they soon provide a heuristic function that has not been given but has been learned in the sense of [121].

The difference between the heuristic-search approaches such as LRTDP and ILAO* on the one hand, and classical DP methods like VI on the other, is more prominent when an informative heuristic like h_{min} is used to seed the value function. The three bottom rows of Table 6.2 shows the convergence times for VI, ILAO* and LRTDP with the initial values obtained with the heuristic h_{min} but with the time spent computing such heuristic values excluded. An average of such times is displayed in the last column of Table 6.1, as these times are roughly equal for the different

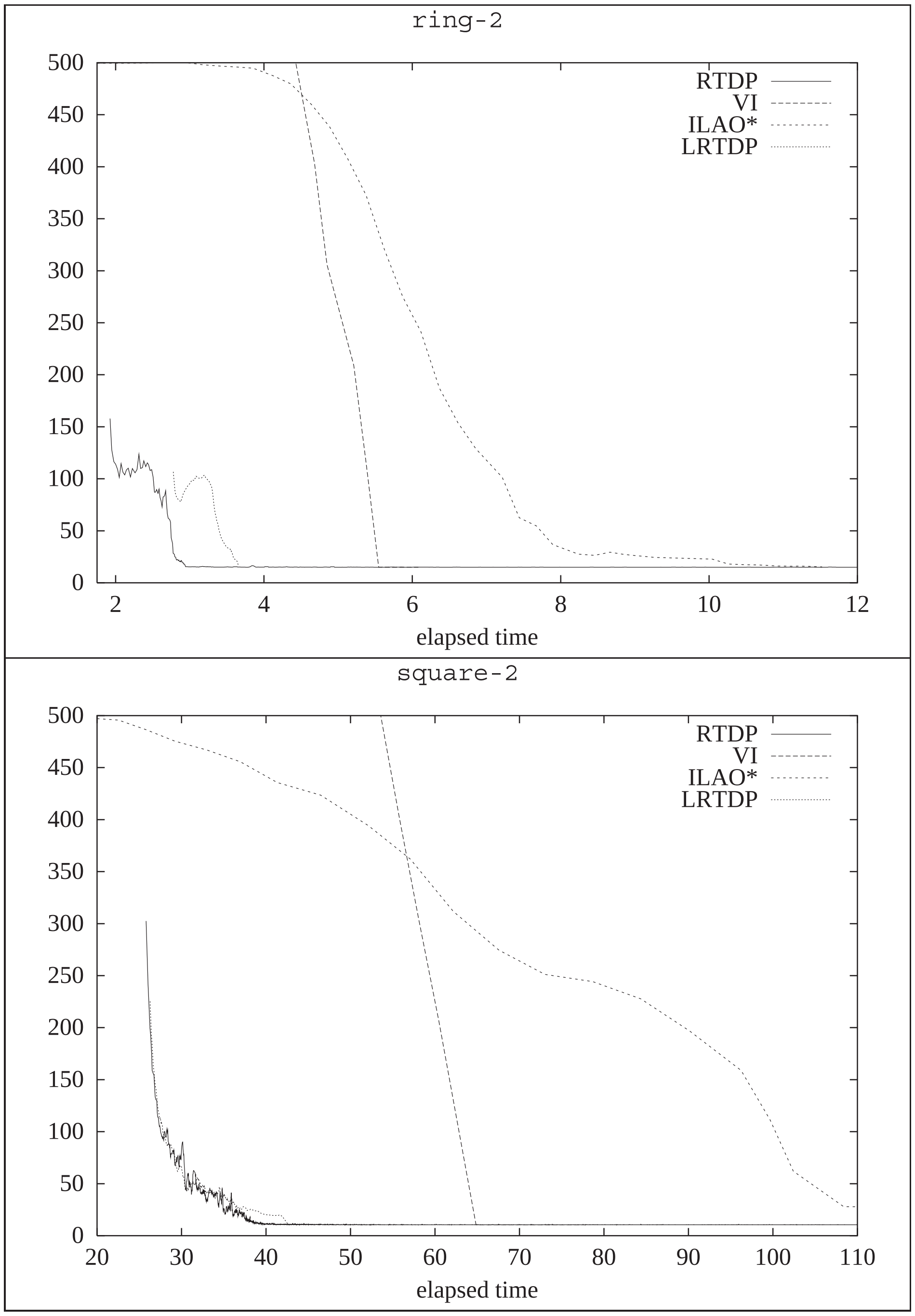


Figure 6.4: Quality profiles: Average cost to the goal vs. time for RTDP, VI, ILAO* and LRTDP with the heuristic $h = 0$ and $\epsilon = 10^{-3}$.

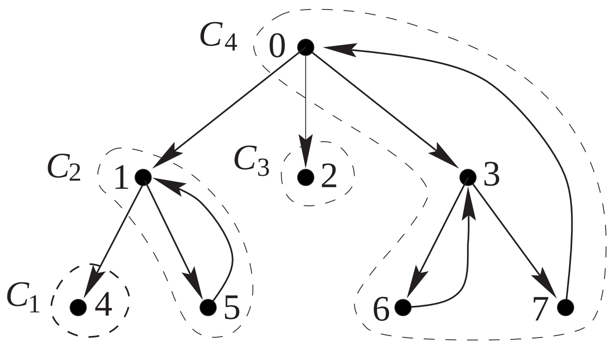


Figure 6.5: A graph and its strongly-connected components.

methods (the standard deviation is around 5% of the average time in the worst case). Clearly, ILAO* and LRTDP make use of the heuristic information much more effectively than VI, and while the computation of the heuristic values is expensive, LRTDP with h_{min}, taking into account the time to compute the heuristic values, remains competitive with VI with $h = 0$. E.g., for `square-2` the overall time for the former is $0.321 + 59.671 = 59.992$ seconds while the time for the latter is 84.301 seconds. Something similar occurs in `square-1`, `y-2` and others. Of course, these results could be improved by speeding up the computation of the heuristic h_{min}.

6.5.3 Heuristic Dynamic Programming

Another instance of the general FIND-and-REVISE schema can be considered in which the FIND operation is carried out by a systematic Depth-First Search that keeps track of the states visited. In addition, a *labeling scheme* is implemented on top of this search that detects, with almost no *overhead*, when a state is *solved*, and hence, when it can be skipped *in all future searches*. As in the LRTDP case, a state s is defined as *solved* when the value function J is ϵ-consistent over s and over all states reachable from s using the greedy policy μ_J. Clearly, when this condition holds no further updates are needed in s or the states reachable from s. The resulting algorithm terminates when the initial state is labeled as solved and hence when an ϵ-consistent value function has been obtained.

As before, due to the presence of cycles in the greedy graph, bottom-up algorithms common in AO* implementations cannot be used. Indeed, if s is reachable (in the greedy graph) from a descendant s' of s, then bottom-up approaches will be unable to label either state as solved. In the LRTDP case, the procedure CHECKSOLVED performs a search to detect when a given state can be labeled as solved. This idea can be improved by removing the need for an extra search for label checking. The label checking will be done as part of the FIND (DFS) search with almost no overhead, exploiting Tarjan's linear algorithm for detecting the *strongly-connected components* of a directed graph [191], and a correspondence between the strongly-connected components of the greedy graph and the minimal collections of states that can be labeled at the same time.

Consider the (directed) greedy graph G_J and the relation $\sim$ between pairs of states s and s' that holds when $s = s'$ or when s is reachable from s' and s' is reachable from s in G_J. The strongly-connected components of G_J are the equivalence classes defined by this relation and form a partition of the set of states in G_J [78]. For example, for the greedy graph in Fig. 6.5, where 2 and 4 are terminal (goal) states, the strongly-connected components are $C_1 = \{4\}$, $C_2 = \{1, 5\}$, $C_3 = \{2\}$, and $C_4 = \{0, 3, 6, 7\}$. Tarjan's algorithm detects the strongly-connected components of a directed graph in time $O(n + e)$ *while traversing the graph depth-first*, where n stands for the number of states ($n \leq |S|$ in G_J) and e for the number of edges.

The relationship between labeling and strongly-connected components in G_J is direct. Let us say that a *component C is ϵ-consistent* when all states $s \in C$ are ϵ-consistent, and that a *component C is solved* when every state $s \in C$ is solved. Then, define G_J^C as the graph whose nodes are the components of G_J and whose directed edges are $C \to C'$ when some state in C' is reachable from some state in C. Clearly, G_J^C is an *acyclic graph* as two components which are reachable from each other will be collapsed into the same equivalence class. In addition,

algorithm	`small-b`	`large-b`	`h-track`	`y-1`	`y-2`
VI$(h=0)$	1.098	**3.942**	19.718	16.266	62.361
ILAO*$(h=0)$	2.524	12.683	45.068	58.487	187.054
LRTDP$(h=0)$	**0.875**	6.057	**15.277**	**10.083**	**34.984**
HDP$(h=0)$	1.230	7.474	35.495	10.707	47.245
VI(h_{min})	1.032	3.478	11.764	15.064	55.864
ILAO*(h_{min})	0.575	2.882	14.631	0.538	1.124
LRTDP(h_{min})	**0.385**	2.140	7.143	0.239	0.492
HDP(h_{min})	0.536	**1.128**	**4.301**	**0.190**	**0.400**

Table 6.3: Convergence time in seconds for the value iteration, ILAO*, LRTDP, and HDP algorithms with the initial value functions $h = 0$ and h_{min} and $\epsilon = 10^{-3}$ for several racetrack instances with parameter $p = 0.1$. Times for RTDP are not shown as they exceed the cutoff time for convergence (10 minutes). All times are in seconds, and the fastest ones are shown in bold.

1. a state s is solved iff its component C is solved, and furthermore,
2. a component C is solved iff C is consistent and all components C', $C \to C'$, are solved.

The problem of *labeling states in the cyclic graph G_J* can thus be mapped into the problem of *labeling the components in the acyclic graph G_J^C*, which can be done in bottom up fashion.

From Fig. 6.5 is easy to visualize the component graph associated with the greedy graph. Thus, if 2 is the only inconsistent state, for example, the components C_1 and C_2 can be labeled as solved, while leaving C_3 and C_4 unsolved.

Alg. 13 shows the code for the overall algorithm that is called *Heuristic Dynamic Programming* or HDP. HDP works by repeatedly calling SCC-DFS(s, ϵ) until the state s becomes solved. The routine SCC-DFS uses two stacks that need to be initialized as empty each time the routine is called.

SCC-DFS is a variation of Tarjan's algorithm for detecting strongly-connected components that are ϵ consistent. For each such component, all states in it are labeled as solved since their values must have converged. Whenever an inconsistent state is found, the component discovering phase of the algorithm is stopped, and an update of states in a depth-first transversal of the greedy graph is triggered. Such transversal never proceed beneath inconsistent states.

As shown in Alg. 14, Tarjan's algorithm uses two indices per node which are the visit number S.IDX and the 'low-link' number S.LOW; two nodes with the same low-link number belong to the same component. The boolean variable $flag$ and the (normal) propagation of the visit numbers prevent a component from being labeled as solved when it is inconsistent or can reach an inconsistent component.

HDP inherits its convergence and optimality properties from the FIND-and-REVISE schema. Its correctness follows directly from the correctness of Tarjan's algorithm [191], and the labeling mechanism and is summarized by

Theorem 37 *The value function computed by* HDP *for a planning model S1–S7 under assumptions A0–A2, given an initial admissible and monotonic value function, is ϵ-consistent.*

Empirical Evaluation

The performance of the HDP algorithm is evaluated in comparison with the other Heuristic Search/DP algorithms ILAO* and LRTDP. As before, value iteration is used to set a baseline.

The first experiment in Table 6.3 shows the convergence times for the first 5 instances of the racetrack domain with the parameter $p = 0.1$ and $\epsilon = 10^{-3}$ for the h_{min} and $h = 0$ heuristic functions.

As it can be seen, HDP dominates, except for the smallest problem, all other algorithms for the case of the h_{min} heuristic. In the case of the $h = 0$ heuristic, the LRTDP algorithm continues to dominate, a fact that can be explained as before.

These preliminary results are further explored with additional experiments over bigger instances of the problems and with different heuristics. The results are shown in Table 6.4. This time the different parameter $p = 0.2$ is used

algorithm	`square-1`	`square-2`	`ring-1`	`ring-2`	`ring-3`	`ring-4`
$\lvert S\rvert$	42071	383950	5895	33068	94369	353991
$J^*(s_0)$	8.07350	11.13041	11.04923	16.09833	22.04798	27.78572
% relevant	1.79	0.25	10.70	11.10	10.91	9.98
$h_{min}(s_0)$	7.0	10.0	10.0	14.0	19.0	24.0
time for h_{min}	4.509	72.853	0.555	4.558	16.745	85.766
VI(h_{min})	5.873	81.270	0.614	5.287	19.466	90.895
ILAO*(h_{min})	0.389	0.942	0.332	6.095	24.422	145.047
LRTDP(h_{min})	0.179	0.363	0.143	1.301	4.928	37.992
HDP(h_{min})	**0.148**	**0.275**	**0.121**	**1.101**	**4.145**	**30.511**
VI$(h_{min}/2)$	6.147	89.248	0.636	5.704	24.385	100.130
ILAO*$(h_{min}/2)$	2.106	68.028	0.765	9.801	32.415	174.089
LRTDP$(h_{min}/2)$	**0.813**	**7.736**	**0.302**	**2.480**	**12.687**	152.200
HDP$(h_{min}/2)$	1.255	20.952	0.331	3.536	15.982	**98.033**
VI$(h = 0)$	8.445	89.152	0.675	5.809	20.979	**101.616**
ILAO*$(h = 0)$	11.230	187.463	1.290	9.894	45.989	310.876
LRTDP$(h = 0)$	**3.286**	**45.514**	**0.431**	**3.682**	**19.424**	261.286
HDP$(h = 0)$	3.579	79.171	0.994	6.173	25.992	157.646

Table 6.4: Problem data and convergence time in seconds for the different algorithms with different heuristics. Results for $\epsilon = 10^{-3}$ and probability $p = 0.2$. All times are in seconds, and the fastest ones are shown in bold.

to see if it has a sensible effect on the algorithms. The problem data for the different instances is shown in the first rows of the tables. The experiments are carried out with the three heuristics: $h = 0$, h_{min}, and $h_{min}/2$ whose time to compute is roughly equal in the latter two cases.

As is shown in Table 6.4, HDP dominates the other algorithms over all the instances for $h = h_{min}$, while LRTDP is best (with one or two exceptions) when the weaker heuristics $h_{min}/2$ and 0 are used. Thus, while HDP seems best for exploiting good heuristic information over these instances, LRTDP bootstraps more quickly (i.e. it quickly computes a good value function). Understanding these differences of performance is part of future work.

6.6 Summary and Notes

This chapter presents the value iteration algorithm for MDPs, that is perhaps the most general and important algorithm for such models. Another important algorithm that is not covered here is the policy iteration algorithm. Although, policy iteration terminates in a finite number of steps, value iteration is generally faster.

Value iteration computes complete policies and hence can be inefficient when only a partial policy closed with respect to a given initial state is sought. The RTDP algorithm was one of the first algorithms for computing partial policies yet its convergence is only asymptotically. A more recent algorithm for partial policy is the LAO* algorithm of [99] which terminates in finite time.

We also introduced new algorithms based on heuristic search for computing partial optimal policies. They are the general FIND-and-REVISE algorithm, the LRTDP algorithm, and the HDP algorithm. FIND-and-REVISE and HDP appeared for the first time in [31], while LRTDP is from [32].

All these algorithms provide initial ideas and methods based on heuristic search for solving MDP problems. We hope that these new ideas motivate the research towards the development of newer algorithms capable of addressing more complex problems that are beyond the scope of the current methods.

We also presented new theoretical results about value functions and the RTDP algorithm. In particular, Theorem 22 shows that it is sufficient to focus in computing value functions with a given small residual. Theorem 25, on the other hand, provides a new proof for the convergence of RTDP, while Theorem 26 bounds the convergence time of each RTDP trial. These results constitute relevant theoretical contributions to the field.

Another related direction of research is the design of admissible and monotonic heuristics to use in the algorithms based on heuristic search. Some proposals are given in Ch. 8.

```
$state$ :: GREEDYACTION()
begin
    return $\operatorname{argmin}_{a \in A(s)} \; g(s,a) + \alpha \sum_{s' \in S} p(s,a,s')\, s'$.VALUE;
end

$state$ :: UPDATE()
begin
    $a$ := $s$.GREEDYACTION();
    $s$.VALUE := $g(s,a) + \alpha \sum_{s' \in S} p(s,a,s')\, s'$.VALUE;
end

$state$ :: PICKNEXTSTATE($a$ : $action$)
begin
    pick $s'$ with probability $p(s,a,s')$;
    return $s'$;
end

$state$ :: RESIDUAL()
begin
    $a$ := $s$.GREEDYACTION();
    return $\big|s$.VALUE $- \big(g(s,a) + \alpha \sum_{s' \in S} p(s,a,s')\, s'$.VALUE$\big)\big|$;
end
```

Algorithm 7: Basic state routines.

```
Start with a lower bound function $J = h$;
repeat
    FIND a state $s$ in the greedy graph $G_J$ with residual $> \epsilon$;
    REVISE $J$ at $s$;
until no such state is found;
return $J$;
```

Algorithm 8: FIND-and-REVISE general schema. The FIND and REVISE procedures are two parameters of FIND-and-REVISE.

```
LAO*($s : state, \epsilon : float$)
begin
    // initialize
    $J := h$;
    if $\neg s$.GOAL() then
        $nodes := \emptyset$;
        $tips := \{s\}$;
    else
        $nodes := \{s\}$;
        $tips := \emptyset$;

    // expand graph until no more tip nodes
1   while $tips \neq \emptyset$ do
        $s'$ := some node in $tips$;
        $a := s'$.GREEDYACTION();
        foreach $s'$ such that $p(s', a, s'') > 0$ do
2           $nodes := nodes \cup \{ s'' \}$;

        // update values, recompute graph and tip nodes
        Perform value iteration over the set of states $nodes \cup tips$
        until $J$ has residual $< \epsilon$.

        Compute new sets $nodes'$ and $tips'$ by performing a depth-first
        visit of $G_J$ starting at $s$. A state $s$ is inserted in $nodes'$ iff all
        its successors in $G_J$ are in $nodes$; otherwise it is inserted in $tips'$.
        The search does not proceed beneath the states in $nodes \cup tips$.

        $nodes := nodes'$;
        $tips := tips'$;

    // convergence test
    if residual over $nodes$ is $> \epsilon$ then goto line marked 1;
end
```

Algorithm 9: Pseudo-code for LAO*.

```
ILAO*($s : state, \epsilon : float$)
begin
    // initialize
    $J := h$;
    if $\neg s$.GOAL() then
        $nodes := \emptyset$;
        $tips := \{s\}$;
    else
        $nodes := \{s\}$;
        $tips := \emptyset$;

    // transverse graph and update values until no more tip nodes
1   while $tips \neq \emptyset$ do
        Perform a depth-first transversal of the greedy graph applying DP
        updates in post order. The search does no proceed beneath states
        in $nodes \cup tips$. Also, new sets of $nodes$ and $tips$ are computed
        during the transversal.

    // convergence test
    Perform value iteration over states in $nodes$ until residual $< \epsilon$ or
    the greedy graph changes. In the latter case, goto line marked 1.
end
```

Algorithm 10: Pseudo-code for ILAO*.

```
CHECKSOLVED($s : state, \epsilon : float$)
begin
    // initialization
    $rv := true$;
    $open :=$ EMPTYSTACK;
    $closed :=$ EMPTYSTACK;
    if $\neg s$.SOLVED then $open$.PUSH($s$);
    // main loop
    while $open \neq$ EMPTYSTACK do
        $s := open$.POP();
        $closed$.PUSH($s$);
        // check residual
        if $s$.RESIDUAL() $> \epsilon$ then
            $rv := false$;
            continue
        // expand state and insert successors in open list
        $a := s$.GREEDYACTION();
        foreach $s'$ such that $p(s, a, s') > 0$ do
            if $\neg s'$.SOLVED & $\neg$IN($s', open \cup closed$)
                $open$.PUSH($s'$);
    // either label relevant states or update states
    if $rv := true$ then
        // label relevant states
        foreach $s' \in closed$ do
            $s'$.SOLVED $:= true$;
    else
        // update states with residuals and ancestors
        while $closed \neq$ EMPTYSTACK do
            $s := closed$.POP();
            $s$.UPDATE();
    return $rv$;
end
```

Algorithm 11: CHECKSOLVED procedure.

```
LRTDP($s$ : $state$, $\epsilon$ : $float$)
begin
    while $\neg s$.SOLVED do LRTDPTRIAL($s, \epsilon$);
end

LRTDPTRIAL($s$ : $state$, $\epsilon$ : $float$)
begin
    // initialization
    $visited$ := EMPTYSTACK;
    // main loop
    while $\neg s$.SOLVED do
        // insert into visited
        $visited$.PUSH($s$);
        // check termination at goal states
        if $s$.GOAL() then break ;
        // pick best action and update hash
        $a$ := $s$.GREEDYACTION();
        $s$.UPDATE();
        // stochastically simulate next state
        $s$ := $s$.PICKNEXTSTATE($a$);
    // try labeling visited states in reverse order
    while $visited \neq$ EMPTYSTACK do
        $s$ := $visited$.POP();
        if $\neg$CHECKSOLVED($s, \epsilon$) then break
end
```

Algorithm 12: LRTDP algorithm.

```
HDP($s : state, \epsilon : float$)
begin
    // initialization
    $stack$ := EMPTYSTACK;
    $visited$ := EMPTYSTACK;

    // main loop
    while $\neg s$.SOLVED do
        // perform DFS to detect SCC's
        $index := 0$;
        SCC-DFS($s, \epsilon$);

        // reset IDX to $\infty$ for visited states
        while $\neg visited$.EMPTY() do
            $s := visited$.POP();
            $s$.IDX $:= \infty$;

        // clean stack
        while $\neg stack$.EMPTY() do
            $stack$.POP();
end
```

Algorithm 13: HDP algorithm.

```
SCC-DFS($s : state, \epsilon : float$)
begin
    // base cases
    if $s$.SOLVED $\vee$ $s$.GOAL() then
        $s$.SOLVED := $true$;
        return $false$;
    if $s$.RESIDUAL() $> \epsilon$ then
        $s$.UPDATE();
        return $true$;
    // mark state as active
    $visited$.PUSH($s$);
    $stack$.PUSH($s$);
    $s$.IDX := $s$.LOW := $index$;
    $index := index + 1$;
    // expand node and make recursive call
    $flag := false$;
    $a := s$.GREEDYACTION();
    foreach $s'$ such that $p(s, a, s') > 0$ do
        if $s'$.IDX $= \infty$ then
            $flag := flag \vee$ SCC-DFS($s', \epsilon$);
            $s$.LOW := min{ $s$.LOW, $s'$.LOW };
        else if $s' \in stack$ then
            $s$.LOW := min{ $s$.LOW, $s'$.IDX };
    // update if necessary, otherwise try to label
    if $flag$ then
        $s$.UPDATE();
        return $true$;
    else if $s$.IDX $=$ $s$.LOW then
        while $stack$.TOP() $\neq s$ do
            $s' := stack$.POP();
            $s'$.SOLVED := $true$;
        $stack$.POP();
        $s$.SOLVED := $true$;
    return $flag$;
end
```

Algorithm 14: SCC-DFS procedure.

Chapter 7

Algorithms for POMDPs

In this chapter, we present different algorithms and new theoretical results for POMDPs. On the side of the algorithms, we review the important class of algorithms that are based on an result by Sondik [186], and the algorithms that are based on discretizations of the belief space. For the latter class, we have developed new optimal algorithms that show better performance on the benchmarks than the best known algorithm based on Sondik's result.

On the theoretical side, we use provide a novel definition of robustness for algorithm based on discretizations, and using the strong continuity properties shown in Ch. 3, we present a new algorithm that is optimal and robust. Up to my knowledge, this is the first algorithm based on discretizations that offers such strong guarantees. This algorithm appeared first in [23].

The algorithms considered in this chapter are those for decision problems under partial observability which correspond to either non-deterministic or stochastic POMDPs. As before, only algorithms for the stochastic case will be presented since it is obvious how to obtain the corresponding non-deterministic version of the algorithms.

Exact algorithms for the general (discounted) POMDP setting are presented first, and then particular algorithms for the case of Stochastic Shortest-Path problems in belief space (see Ch. 3). The latter class of algorithms is the most relevant to general planning tasks since them correspond to SSP problems in belief space.

Remember that in the case of MDPs, an optimal algorithm is an algorithm that returns a value function that *guarantees* a given residual, or as in the discounted case, a bound on the suboptimality of the resulting policy. The same qualification applies here, so an algorithm for POMDPs is called optimal if it returns a value function (or representation of it) together with a guarantee on the residual, which in the the discounted case, is sufficient to guarantee a bound on the suboptimality of the resulting policy.

The case of general algorithms for POMDPs can be further divided into algorithms that work with *implicit* value functions or those that work with *explicit* value functions. In any case, there is always a method for computing the value for a belief state x, and so a greedy policy with respect to such a value function can be computed. The most studied algorithms that work with implicit value functions are those based on Sondik's representation theorem [186, 205], while the most studied algorithms that work with explicit functions are those known as grid-based algorithms [104].

7.1 Algorithms based on Sondik's Representation

Throughout this chapter, a stochastic transition system in belief space like SB1–SB7 of Ch. 3 is assumed. The corresponding Bellman's operators, denoted by T_μ and T, are defined in (3.20) and (3.21) respectively. As was shown in Ch. 3, $T^k J$ converges uniformly to J^* in the discounted case for any initial function J. The following is Sondik's representation

Theorem 38 ([186]) *Assume a model like SB1–SB7, and let J be any piece-wise linear and concave function. Then,*

$T^k J$ is piece-wise linear and concave, and

$$(T^k J)(x) \;=\; \min\{\,(x,\gamma)\,:\,\gamma\in\Gamma_k\,\}\,; \tag{7.1}$$

where Γ_k is a finite *set of $|S|$-dimensional real vectors and (x,γ) is the inner-product of x and γ [A18].*

Remark: The fact that T^k is piece-wise linear and concave does not imply that J^* is piece-wise linear and concave.

Therefore, a finite set of vectors is enough to implicitly represent a value function with an infinite domain. Algorithms that work with Sondik's representation theorem define methods to compute the sequence of sets $\{\Gamma_k\}_k$ as efficiently as possible. Then, after stopping at some k large enough to guarantee a given desired precision, the approximation $T^k J$ is recovered by means of (7.1) and a greedy policy is thus defined.

In this section, the exact incremental pruning algorithm (IP) of [204] and the approximated Point-Based Value Iteration algorithm (PB-VI) of [205] are presented. The former has been shown to be the most efficient known exact algorithm based on Sondik's representation theorem both in theory and practice [44]. The latter, on the other hand, is a recent approximation algorithm that has shown better performance than IP.

Other algorithms based on Sondik's theorem are Sondik's one-pass algorithm [186], Cheng's linear support and relaxed region [47], Monahan's and Eagle's algorithms [150, 73], and the Witness algorithm [113].

7.1.1 Incremental Pruning and Point-Based Value Iteration

In lieu of Theorem 38, Γ and Λ will be used to denote set of vectors, $\Gamma(x)$ to denote the value function defined as

$$\Gamma(x)\;\stackrel{\text{def}}{=}\;\min\{\,(x,\gamma)\,:\,\gamma\in\Gamma\,\}\,,$$

and $T\Gamma$ to denote both the result of applying T to the function defined by Γ and the set of vectors representing it.

A vector $\gamma\in\Gamma$ is *redundant* if Γ and $\Gamma\setminus\{\gamma\}$ define the same function; a non-redundant vector is called an *essential* vector. Obviously, γ is essential iff there exists a belief state x such that $(x,\gamma)<(\Gamma\setminus\{\gamma\})(x)$. The collection of beliefs that testify to the essential character of γ are called *witnesses* for γ, and are denoted with $\omega(\gamma,\Gamma)$ or just $\omega(\gamma)$ when there is no risk of confusion.

Since essential vectors are the important ones, a function defined by Γ can be uniquely represented as the set of its essential vectors $ess(\Gamma)$. This fact justifies the use of the adjective 'the' in the last part of the first paragraph of this section.

Incremental pruning is an algorithm for finding $ess(T\Gamma)$ from $ess(\Gamma)$ and thus enabling the value iteration algorithm.

In order to describe the algorithm, the following operations between set of vectors and functions are considered. Let Γ and Λ be sets of vectors and f a real-valued function defined on $S\times S$. The sum $\Gamma\oplus\Lambda$ is defined as the collection of functions $s\rightsquigarrow\gamma(s)+\lambda(s)$ for $\gamma\in\Gamma$ and $\lambda\in\Lambda$, the scalar multiplication $\alpha\Gamma$ is defined as the collection of functions $s\rightsquigarrow\alpha\gamma(s)$, and the convolution $\Gamma * f$ as the collection of functions $\gamma * f$ (the convolution of γ and f) given by

$$(\gamma * f)(s)\;\stackrel{\text{def}}{=}\;\sum_{s'\in S}\gamma(s')f(s',s)\,.$$

A direct consequence of the proof of Sondik's theorem is the following: if Γ is a collection of vectors representing the value function J, then TJ is represented by the essential vectors of

$$T\Gamma \;=\; \bigcup_{a\in A}\left\{\,G_a \;+\; \alpha\bigoplus_{o\in O}(\Gamma * F_{a,o})\right\} \tag{7.2}$$

where $G_a(s)\stackrel{\text{def}}{=}g(s,a)$ and $F_{a,o}(s',s)\stackrel{\text{def}}{=}p(s,a,s')q(s',a,o)$.

The incremental pruning algorithm computes the essential vectors $ess(T\Gamma)$ incrementally using the expression (7.2). For any two sets Γ and Λ, the set $ess(\Gamma\oplus\Lambda)$ can be computed with the procedure ESS-SUM in Alg. 15. The test $\omega(\gamma,\Gamma)\cup\omega(\lambda,\Lambda)\neq\emptyset$ in the third line is performed by solving the linear programming problem defined as

maximize β subject to

$$\begin{array}{rcll} (\gamma,x) & \geq & \beta + (\gamma',x) & \gamma' \in \Gamma,\ \gamma' \neq \gamma, \\ (\lambda,x) & \geq & \beta + (\lambda',x) & \lambda' \in \Lambda,\ \lambda' \neq \lambda, \\ x(s) & \geq & 0 & s \in S, \\ \sum_{s\in S} x(s) & = & 1 & \end{array}$$

with the $|S|+1$ decision variables given by β and the $x(s)$'s. It can be shown that this problem is always feasible, and that $\omega(\gamma,\Gamma)$ intersects $\omega(\lambda,\Lambda)$ if and only if the solution for β is positive.

The basic routine ESS-SUM is used to define the IP routine for computing the essential vectors of a sum of sets $\oplus_{i=1}^{n}\Gamma_i$ as shown in Alg. 15. Finally, the DP update is implemented using expression (7.2) in the routine IP-DP-UPDATE also shown in Alg. 15.

The main bottleneck in the IP algorithm is the large number of LP programs that need to be solved, a number that often grows exponentially in the number of observations. The point-based value iteration algorithm alleviates this explosion by sacrificing optimality. For a set Γ of essential vectors representing a value function, then a PB-VI update compute a set of essential vectors approximating $T\Gamma$ by solving at most $|\Gamma| + |T\Gamma|$ LP problems, which is considerably smaller than IP. The details of PB-VI can be found in [205].

7.2 Algorithms Based on Grid Discretizations

The basic idea behind a grid-based algorithm is to find the optimal value function over a finite number of belief states and then use such values to approximate the value for other belief states.

This intuitive idea is now formalized. Define as a *grid* G over the belief space B any *finite collection* of beliefs together with a *projection* map $\eta : B \to G$ from the belief space to the grid. A *grid-based approximation* to J^* is a real-valued vector $\tilde{J} : G \to \mathbb{R}$ that is used to approximate $J^*(x)$ as $\tilde{J}(\eta(x))$. Thus, the following definition seems appropriate. A *grid-based algorithm* is an algorithm that on input G, and possibly other parameters, outputs a grid-based approximation $\tilde{J}$. Observe that there are no restrictions about the conditions that the approximation needs to satisfy; yet some conditions will be imposed later. As a remark, other notions of grid-based algorithms had been given; see [104] for a survey of exact and approximate POMDP algorithms.

Several grid-based algorithms had been proposed for finding approximate solutions to POMDPs (e.g. [136, 40, 24, 104]) but, although some of them have shown good performance over benchmark problems, none of them offer bounds on the quality of the results. That is, a bound over the residual, or approximation error,

$$\|J^* - \tilde{J} \circ \eta\| \;=\; \sup\,\{\, |J^*(x) - \tilde{J}(\eta(x))| : x \in B \,\}.$$

From now on, it is assumed that the projection η is given in terms of a *distance* (or *metric* [A5]) σ in belief space such that $\eta(x)$ is a nearest grid-point to x under σ (with ties broken in a predefined way). In such a case, the grid is said to be metric and the *mesh* of the grid is defined as the maximum separation between a grid-point and its surrogate points, i.e.

$$\text{mesh}(G) \;\overset{\text{def}}{=}\; \sup_{x\in G}\ \sup\,\{\, \sigma(x,y) : y \in \eta^{-1}(x) \,\}.$$

Using these definitions, a grid-based algorithm is then termed as *robust in the strong sense* if, independently of the position of the grid points, the approximation error goes to zero as the mesh of the grid goes to zero. That is,

Definition 1 *Let G be a metric grid on (B,σ), η the projection induced by G and $\tilde{J}_G : G \to \mathbb{R}$ a grid-based approximation to J^*. Then, $\tilde{J}_G$ is said to be* robust in the strong sense *if*

$$\lim_{\epsilon \searrow 0}\ \sup_{G}\ \|J^* - \tilde{J}_G \circ \eta\,\| \;=\; 0$$

where the supremum is over all grids with mesh$(G) \leq \epsilon$.

```
ESS-SUM$(\Gamma, \Lambda)$
begin
    $\Psi := \emptyset$;
    foreach $\gamma \in \Gamma$ and $\lambda \in \Lambda$ do
        if $\omega(\gamma, \Gamma) \cup \omega(\lambda, \Lambda) \neq \emptyset$ then
            $\Psi := \Psi \cup \{\gamma + \lambda\}$;
    return $\Psi$;
end

IP$(\{\Gamma_i : 1 \leq i \leq n\})$
begin
    $\Psi := \Gamma_1$;
    for $i = 2$ to $n$ do
        $\Psi :=$ ESS-SUM$(\Psi, \Gamma_i)$;
    return $\Psi$;
end

IP-DP-UPDATE$(\Gamma)$
begin
    foreach $a \in A$ do
        foreach $o \in O$ do
            $\Gamma_{a,o} := \alpha(\Gamma * F_{a,o})$;
        $\Lambda_a :=$ IP$(\{\Gamma_{a,o} : o \in O\})$;
    return $ess(\{G_a + \Lambda_a : a \in A\})$;
end
```

Algorithm 15: Routines implementing the IP dynamic programming update for computing the essential vectors $ess(T\Gamma)$.

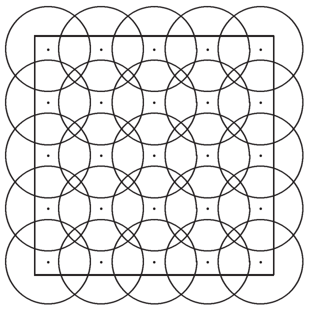


Figure 7.1: An ϵ-cover of the unit square.

In the rest of this section, a grid-based algorithm that is robust in the strong sense is presented together with its optimality guarantees. The algorithm first appeared in [23].

Consider the metric ρ over the belief space defined in (3.22), and a finite collection $\mathcal{B} = \{B_1, \dots, B_m\}$ of open balls with respect to ρ centered at $\{x_1, \dots, x_m\}$ [A6]. Such a collection is called an *ϵ-cover* if each ball is of radius ϵ and their union contains B. For example, Fig. 7.1 shows an ϵ-cover of the unit square. It is not hard to see that an ϵ-cover always exists for every $\epsilon > 0$ and that the x_i can be chosen so that $x_i(j)$ are rational for $1 \leq i \leq m, 1 \leq j \leq |S|$. Indeed, the fact that a finite cover always exists for positive ϵ is known as the space (B, ρ) being totally bounded [A10], and the fact that all centers can be chosen with rational coordinates follows from separability [A9].

A cover like $\mathcal{B}$ induces a projection $\eta : B \to \{x_1, \dots, x_m\}$ given by $\eta(x) = x_i$ if and only if i is minimum such that $x \in B_i$. Then, by Theorem 14,

$$|J^*(x) - J^*(\eta(x))| \leq \frac{\overline{g}}{1-\alpha}\rho(x, \eta(x))^\gamma \leq \frac{\overline{g}\epsilon^\gamma}{1-\alpha}.$$

So, for obtaining a robust grid-based algorithm it is enough to construct a 'good' approximation to $J^* \circ \eta$.

Consider the sequence of real-valued vectors $\{\tilde{J}_k\}$ over the set of centers $\{x_k\}_k$ defined as

$$\tilde{J}_0(x_i) \overset{\text{def}}{=} 0, \tag{7.3}$$

$$\tilde{J}_{k+1}(x_i) \overset{\text{def}}{=} \min_{a \in A(x_i)} \left\{ g(x_i, a) + \alpha \sum_{o \in O(x_i, a)} q(x_i, a, o) \tilde{J}_k(\eta((x_i)^o_a)) \right\}. \tag{7.4}$$

This is the value iteration algorithm applied to an MDP with state space given by the centers $\{x_k\}_k$ and transition probabilities

$$p(x_i, a, x_j) \overset{\text{def}}{=} \sum_{o : \eta((x_i)^o_a) = x_j} q(x_i, a, o).$$

It is easy to see that the associated Bellman's operators are identical to the restriction of $T_\mu(J \circ \eta)$ and $T(J \circ \eta)$ to the set $\{x_1, \dots, x_m\}$. Since η^{-1} partitions the belief space into a finite number of pieces, (7.4) defines a grid-based algorithm for POMDPs. The following theorems bound the approximation error and the loss incurred by the greedy policy when the value iteration method $\{\tilde{J}_k \circ \eta\}_k$ is stopped at an appropriate iteration.

Theorem 39 *Assume a model like SB1–SB7 with $\alpha < 1$ and let γ be chosen as in Theorem 14. Let $\tilde{J}_k$ be defined by* (7.4)*, then*

$$\|J^* - \tilde{J}_k \circ \eta\| \leq \frac{\|g\|\epsilon^\gamma}{(1-\alpha)^2} + \frac{\alpha^k \|g\|}{1-\alpha},$$

$$\lim_{k\to\infty} \|J^* - \tilde{J}_k \circ \eta\| \leq \frac{\|g\|\epsilon^\gamma}{(1-\alpha)^2}.$$

Corollary 40 *Assume the same conditions of the theorem, and let μ^k be the greedy policy with respect to $\tilde{J}_k \circ \eta$, i.e. $T_{\mu^k}(\tilde{J}_k \circ \eta) = T(\tilde{J}_k \circ \eta)$. Then,*

$$\|J^* - J_{\mu^k}\| \;\leq\; \frac{2\alpha(1+\alpha)\|g\|\epsilon^\gamma}{(1-\alpha)^3}$$

for every $k \geq k_0$ where k_0 is such that $\alpha^{k_0} \leq \epsilon^\gamma/(1-\alpha)$.

These two results show the correctness of the grid-based algorithm shown in Alg. 16. The algorithm is robust in the strong sense, tractable in the parameters ϵ^{-1}, α, $|O|$, and only exponential in the dimension $|S|$. The exponentiality in $|S|$ cannot be removed since it is known that solving general POMDPs, either exactly or approximately, is NP-HARD [138].

Although the algorithm is correct and tractable in all parameters except the dimensionality, its applicability is still questionable since the state space of the discretization grows as fast as $|S|^{\epsilon^{-1}}$ which is clearly impractical. Thus, the above algorithm is only of theoretical interest. However, the ideas developed for this algorithm can be applied to the case of stochastic shortest-path problems within the FIND-and-REVISE framework of the previous chapter.

7.2.1 Proofs

Proof of Theorem 39: First note that

$$\begin{aligned}|J^*(x) - \tilde{J}_k(\eta(x))| &= |J^*(x) - J^*(\eta(x)) + J^*(\eta(x)) - \tilde{J}_k(\eta(x))| \\ &\leq \frac{\|g\|\epsilon^\gamma}{1-\alpha} + |J^*(\eta(x)) - \tilde{J}_k(\eta(x))|.\end{aligned}$$

Now, use induction on k to find a bound on $|J^*(x) - \tilde{J}_k(x)|$ for $x \in \{x_1, \ldots, x_m\}$ as follows

$$\begin{aligned}|J^*(x) - \tilde{J}_1(x)| &\leq \alpha \sum_{o\in O(x,a)} q(x,a,o)\,\big|J^*(x_a^o) - \tilde{J}_0(\eta(x_a^o))\big| \;\leq\; \alpha\|J^*\|, \\ |J^*(x) - \tilde{J}_2(x)| &\leq \alpha \sum_{o\in O(x,a)} q(x,a,o)\,\big|J^*(x_a^o) - \tilde{J}_1(\eta(x_a^o))\big| \\ &\leq \alpha \sum_{o\in O(x,a)} q(x,a,o)\left(\frac{\|g\|\epsilon^\gamma}{1-\alpha} + \alpha\|J^*\|\right) \\ &= \frac{\|g\|\epsilon^\gamma}{1-\alpha}\alpha + \alpha^2\|J^*\|\end{aligned}$$

where $a \in A(x)$ (see the proof of Theorem 14). Then,

$$\begin{aligned}|J^*(x) - \tilde{J}_{k+1}(x)| &\leq \alpha \sum_{o\in O(x,a)} q(x,a,o)\,\big|J^*(x_a^o) - \tilde{J}_k(\eta(x_a^o))\big| \\ &\leq \alpha\left(\frac{\|g\|\epsilon^\gamma}{1-\alpha} + \frac{\|g\|\epsilon^\gamma}{1-\alpha}\sum_{j=1}^{k-1}\alpha^j \;+\; \alpha^k\|J^*\|\right) \\ &= \frac{\|g\|\epsilon^\gamma}{1-\alpha}\sum_{j=1}^{k}\alpha^j \;+\; \alpha^{k+1}\|J^*\|.\end{aligned}$$

Therefore,

$$\sup_{x\in B}|J^*(x) - \tilde{J}_k(\eta(x))| \;\leq\; \frac{\|g\|\epsilon^\gamma}{1-\alpha} + \frac{\|g\|\epsilon^\gamma}{1-\alpha}\sum_{j=1}^{k-1}\alpha^j \;+\; \alpha^k\|J^*\| \;\leq\; \frac{\|g\|\epsilon^\gamma}{(1-\alpha)^2} + \frac{\alpha^k\|g\|}{1-\alpha}.$$

□

Proof of Corollary 40: Note that

$$
\begin{aligned}
&|(T(\tilde{J}_k \circ \eta))(x) - \tilde{J}_k(\eta(x))| \\
&\quad \le\ |(T(\tilde{J}_k \circ \eta))(x) - (TJ^*)(x)|\ +\ |J^*(x) - \tilde{J}_k(\eta(x))| \\
&\quad \le\ \Bigg| \min_{a\in A(x)} \Big\{ g(x,a) + \alpha \sum_{o\in O(x,a)} q(x,a,o)\, \tilde{J}_k(\eta(x_a^o)) \Big\}\ - \\
&\qquad \min_{a\in A(x)} \Big\{ g(x,a) + \alpha \sum_{o\in O(x,a)} q(x,a,o)\, J^*(x_a^o) \Big\} \Bigg|\ +\ |J^*(x) - \tilde{J}_k(\eta(x))| \\
&\quad \le\ \alpha \sum_{o\in O(x,a)} q(x,a,o)\, |\tilde{J}_k(\eta(x_a^o)) - J^*(x_a^o)|\ +\ |J^*(x) - \tilde{J}_k(\eta(x))| \\
&\quad \le\ (1+\alpha)\|J^* - \tilde{J}_k \circ \eta\| \\
&\quad \le\ (1+\alpha)\left(\frac{\|g\|\epsilon^\gamma}{(1-\alpha)^2}\ +\ \frac{\alpha^k\|g\|}{1-\alpha} \right) \\
&\quad \le\ (1+\alpha)\frac{2\|g\|\epsilon^\gamma}{(1-\alpha)^2}
\end{aligned}
$$

for all $k \ge k_0$. Now, use Theorem 5 to find a bound on the suboptimality of μ^k:

$$J_{\mu^k}(x) - J^*(x)\ \le\ \frac{\alpha}{1-\alpha}\left(\sup_{x\in B} |(T(\tilde{J}_k \circ \eta))(x) - \tilde{J}_k(\eta(x))| \right)\ \le\ \frac{\alpha(1+\alpha)}{1-\alpha}\frac{2\|g\|\epsilon^\gamma}{(1-\alpha)^2}$$

for all $k \ge k_0$. □

7.3 Find-and-Revise Algorithms

This section focuses on algorithms for solving stochastic shortest-path problems in belief space. As was shown in Theorem 18, such problems can be solved by using value iteration over the belief-MDP with any initial value function J such that $J(x) = 0$ for all goal beliefs $x \in B_G$.

In order to apply the FIND-and-REVISE algorithms for MDPs, a discretization of the belief-space into a finite number of elements is needed. Theorem 14 shows that if the discretization is such that it 'respects' supports, then the resulting grid-based algorithm is robust in the strong sense.

FIND-and-REVISE algorithms explore the belief space incrementally until a solution is obtained. Thus, a 'procedure' for computing discretizations in an incremental way is needed. The following method had been used in [26, 27] for solving SSP problems in belief space.

The discretization for belief x, that is denoted as $\tilde{x}$ and depends on a parameter L (positive integer), is defined as

$$\tilde{x}(s)\ \overset{\text{def}}{=}\ K(x)[\![1 + L \cdot x(s)]\!]. \tag{7.5}$$

The notation $[\![\xi]\!]$ stands for the closest integer to the real number ξ truncating downwards. The number $K(x)$ is a normalization constant so that $\tilde{x}$ is a proper belief state, i.e. $\sum_{s\in S} \tilde{x}(s) = 1$, while the addition of 1 guarantees that the discretization preserve supports, i.e. $\tilde{x}(s) > 0$ if and only if $x(s) > 0$. From (7.5) is clear that as the parameter L increases, the discretization becomes finer and so $\tilde{x} \to x$ as $L \to \infty$.

Equation (7.5) induces a (finite) discretized version of the belief-MDP that corresponds to the following Bellman operators:

$$(TJ)(\tilde{x})\ \overset{\text{def}}{=}\ \min_{a\in A(\tilde{x})} \left(g(\tilde{x},a)\ +\ \sum_{o\in O(\tilde{x},a)} q(\tilde{x},a,o)J(\widetilde{\tilde{x}_a^o}) \right).$$

data	tiger	cheese	4x4	4x3.CO	4x3	aircraft
$\|S\|$	3	11	16	11	11	12
$\|O\|$	2	7	2	11	6	5
$\|A\|$	3	4	4	4	4	6
$\|x_{init}\|$	2	10	15	9	9	11
α	0.95	0.95	0.95	0.95	0.95	0.95
ϵ	10^{-6}	10^{-6}	10^{-6}	10^{-6}	10^{-6}	10^{-6}

Table 7.1: Information for small POMDP examples in Cassandra's format.

The main benefit of the discretization is that it can be efficiently implemented in time and space linear in the size of the support. Since the supports are preserved by the discretization, the induced MDP is an stochastic shortest-path problem that satisfies assumptions A1–A2 if the original problem satisfies them. Thus, FIND-and-REVISE algorithms can be applied.

7.3.1 Empirical Evaluation

The LRTDP algorithm is evaluated against the Incremental Pruning (IP) and point-based value iteration (PB-VI) algorithms. The former is a FIND-and-REVISE algorithm using above discretization schema while the latter two are algorithms based on Sondik's representation theorem. For LRTDP, different discretization levels of $L = 5, 20, 50, \infty$ are used. The LRTDP experiments were run in a Sun-Fire 280-R with 1Gb of RAM and a clock speed of 750Mhz, while the data for IP and PB-VI are taken from [205] where they were run in a similar machine.

The problems used for the evaluation are the POMDP problems of Tony Cassandra which have become the de-facto standard for empirical evaluation of POMDP algorithms. These problems are specified in Cassandra's POMDP format; both the problems and a description of the format can be found in his web page at `http://www.cs.brown.edu/research/ai/pomd`

The benchmark consists of six problems named `tiger`, `cheese`, `4x4`, `4x3.CO`, `4x3` and `aircraft`. The first problem consists of two doors with a tiger behind one of them, and the task is to decide which door to open after gathering some (imperfect) information [43]. The `cheese` problem is a maze problem with 11 states originally due to McCallum [141]. `4x4` is a navigation grid with 16 states due to Cassandra et al. [43]. `4x3` and `4x3.CO` are Russell and Norvig's 4x3 grid [177] in their original and completely observable version. Finally, `aircraft` is an aircraft identification problem involving 16 states.

Basic information about these problems is shown in Table 7.1, while the results of the experiments are in Table 7.2.

Five quantities are reported for the LRTDP algorithm: the total time to solve the problem, the size of the hash table at termination, the size of the resulting policy, the hash value for the initial belief state at termination ($J(x_{init})$), and an estimate of the quality of the resulting policy ($\widehat{J(x_{init})}$). The size of the policy refers to the number of belief states in the final policy. The value $J(x_{init})$ is always a lower bound for the optimal value $J^*(x_{init})$ *over the discretized problem*, i.e. the MDP problem given in equations (7.3) and (7.4). The estimate $\widehat{J(x_{init})}$ is computed by averaging the discounted accumulated cost of 10,000 trials of operation of the system under the resulting policy. As can be seen in the `tiger` problem, for example, the value $J(x_{init})$ decreases and the quality of the policy increases as L increases. The small gap $|J(x_{init}) - \widehat{J(x_{init})}|$ for $L = \infty$ tells us that the resulting policy is near optimal. Note that solving the problem with $L = \infty$ generates simpler policies than with discretization. This is explained by the fact that the problem has a small number of relevant belief states but not its discretized version.

The bottom of Table 7.2 contains the solution times for the IP and PB-VI algorithms taken from [205] which were run in a similar machine. Results for IP and PB-VI over `4x3` is not reported in [205].

Note that LRTDP dominates both algorithms across all instances and that for the `aircraft` problem the difference is 3 orders of magnitude. The size of these problems is the current limit for algorithms based on Sondik's representation. LRTDP is evaluated over significantly harder problems in Ch. 9.

L	data	`tiger`	`cheese`	`4x4`	`4x3.CO`	`4x3`	`aircraft`
5	time	0.22	0.16	0.16	0.30	3.29	0.56
	|hash|	8	14	40	12	117	14
	|policy|	8	10	7	12	34	12
	$J(x_{init})$	15.1001	4.3406	3.8209	4.5863	5.5061	15.3358
	$\widehat{J(x_{init})}$	5.4749	4.3288	3.7625	4.5608	5.2929	11.8059
20	time	0.22	0.16	0.17	0.28	5.00	1.93
	|hash|	22	14	52	12	362	257
	|policy|	18	10	7	12	90	35
	$J(x_{init})$	7.8050	4.3406	3.8338	4.5863	5.2891	13.8946
	$\widehat{J(x_{init})}$	5.6138	4.3446	3.7709	4.5379	5.0319	11.2972
50	time	1.10	0.16	0.18	0.28	8.19	4.03
	|hash|	38	14	67	12	855	1124
	|policy|	38	10	7	12	202	130
	$J(x_{init})$	6.7101	4.3406	3.7986	4.5863	5.1814	12.7337
	$\widehat{J(x_{init})}$	5.3455	4.3472	3.7634	4.5876	5.0472	11.4376
∞	time	0.16	0.16	0.16	0.29	–	104.69
	|hash|	10	14	37	12	–	7992
	|policy|	8	10	7	12	–	229
	$J(x_{init})$	4.2710	4.3406	3.7666	4.5863	–	11.2404
	$\widehat{J(x_{init})}$	4.2081	4.3471	3.7628	4.5859	–	11.5690
IP	time	79.14	13.90	27.15	3.20	N/A	–
PB-VI	time	0.56	5.00	5.30	2.40	N/A	27,676

Table 7.2: Results for small POMDP examples in Cassandra's format for the LRTDP, IP and PB-VI algorithms. The times are in seconds. A dash for LRTDP means that the problem is not solvable since there are an infinite number of beliefs. N/A means that data for the `4x3` problem is not available for IP and PB-VI, and the dash for IP means the algorithm was unable to solve the `aircraft` problem.

7.4 Summary and Notes

Different algorithms for POMDP models are presented in this chapter. The algorithms can be divided into two classes: those based on Sondik's representation theorem and those based on grid discretizations.

The algorithms in the first class have their origins in the seminal work of [186]. The best known algorithm in this class is the incremental pruning algorithm whose description is based on that found in [203]. Incremental pruning has been shown to be superior both theoretically and in a number of benchmarks to other exact algorithms in this class. An approximation version of IP, known as point-based value iteration, of [205] is also presented. PB-VI shows better performance than IP over benchmark problems.

The algorithms in the second class are based on discretizations of the belief space into a finite number of points. We provide a strong criterion for robustness for these algorithms, and present a novel discretization algorithm that is optimal and robust in such sense. Up to my knowledge, this is the first algorithm with such strong guarantees.

For the class of stochastic-shortest path problems in belief space, the FIND-and-REVISE algorithms of Ch. 6 are combined with a dynamic discretization scheme to obtain algorithms that are optimal and robust.

The different algorithms are evaluated over a number of problems that have become a de-facto benchmark for POMDP problems. The LRTDP instance of FIND-and-REVISE is evaluated against IP and PB-VI. The results show that LRTDP dominates IP and PB-VI over the benchmark. Indeed, this benchmark consists of small problems that are at the boundary of what IP and PB-VI can solve, yet they are relatively easy for LRTDP. In Ch. 9, LRTDP is tested over harder problems.

```
Input: A number $\epsilon > 0$ and a finite collection $\{x_1, \ldots, x_m\}$ so that the open balls of radius $\epsilon$ centered at $x_i$'s
       form a cover for $B$.
Output: An $m$-dimensional real vector $\tilde{J}$ such that
            $\|J^* - \tilde{J} \circ \eta\| \leq \frac{2\|g\|\epsilon^\gamma}{(1-\alpha)^2}$
            $\|J^* - J_{\tilde{\mu}}\| \leq \frac{2\alpha(1+\alpha)\|g\|\epsilon^\gamma}{(1-\alpha)^3}$
        where $\tilde{\mu}$ is the greedy policy with respect to $\tilde{J} \circ \eta$.
begin
    // initialization
    Set $\tilde{J}_0(x_i) = 0$ for $i = 1, \ldots, m$;
    $k \leftarrow 0$;
    // main loop to perform enough DP updates
    while $k \leq (\gamma \log \epsilon - 2\log(1-\alpha))/\log \alpha$ do
        $\tilde{J}_{k+1} \leftarrow T(\tilde{J}_k \circ \eta)$;
        $k \leftarrow k + 1$;
end
```

Algorithm 16: An ϵ-optimal grid-based and strong-robust algorithm for POMDPs.

Chapter 8

Heuristics

This chapter presents heuristic functions for the different planning models. In particular, we present 7 heuristic functions: h^m_{max} for DDP models, two h_{min} functions for non-deterministic and stochastic MDP models respectively, two h_{mdp} functions for non-deterministic and stochastic POMDP models, h^m_{sup} for non-deterministic POMDPs, and h_{sup} for stochastic POMDP. All functions are novel heuristic functions that are admissible and monotonic. The only exception is h^m_{max} for deterministic models that is taken from [102], yet its monotonicity is shown here.

Remember that our main goal is to define domain-independent heuristic functions. Such heuristics need to be extracted from the description of the problem.

Above heuristics have proved to be effective over some models, yet for other models, the design of more informative heuristic is still an active area of research.

Indeed, even in the simple deterministic case, the design of good heuristic function is a main issue. Recent examples of success are the 24-puzzle [126], Rubik's cube [123], and the game of Sokoban [112].

Remember that an admissible heuristic is one that estimates the optimal cost from *below*. In other words, an admissible heuristic is a lower bound for the optimal value function J^*, i.e. h is an admissible heuristic if $h(s) \leq J^*(s)$ for all state s.

The standard approach for obtaining admissible heuristics is the *relaxation method* described in [155]. The general idea is to relax a given problem P into a simpler problem $\tilde{P}$, and then use the optimal value function $\tilde{J}^*$ of $\tilde{P}$ as the heuristic; i.e. $h = \tilde{J}^*$.

The classical example of this method is given in the 15-puzzle where a good heuristic function is obtained by considering a relaxation in which the tiles are allowed to move *independently and without interference*. Thus, the optimal value function, and hence the heuristic, is the sum of the individual Manhattan distances for each tile from their positions to their goal positions. For example, if the current state of the search and goal state correspond respectively to the configurations (a) and (b) in Fig. 8.1, the heuristic function for (a) is equal to 41 while the optimal solution cost is 57.

14	13	15	7
11	12	9	5
6		2	1
4	8	10	3

(a)

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

(b)

Figure 8.1: Two configurations for the 15-puzzle: (a) is a non-goal state, and (b) is the goal state. The Manhattan Distance heuristic value for (a) is 41 and its optimal solution cost is 57.

Of course the success of this method depends on whether the relaxation is non trivial and informative, and whether the relaxation can be solved efficiently.

An efficient and powerful implementation of the relaxation method is obtained with the method of *pattern databases* first defined in [54], and later extended by Korf et al. in [126, 124].

In the 15-puzzle, for example, Korf and Taylor [126] show that instead of adding up estimates for *single* tiles under the no-interference assumption, it is better to consider pairs of tiles (or higher order tuples) and compute the minimum number of moves needed to take each pair of tiles to their goal positions, taking care only of the interferences within each pair. This computation can performed for every pair of tiles and every position, and the results stored into a table in memory which is called a pattern database. Then, during search the heuristic value for a state can be defined as the maximum value of such costs over the pairs compatible with the given state.

It is common practice to use several pattern databases for a given problem, each denoting a different relaxation, and then combine the induced heuristics by taking their maximum. If two pattern databases can be additively combined while preserving admissibility, then the databases are termed *disjoint*; see [124] for details and some experimental results. Finding pattern databases is not always obvious and disjoint ones is even more difficult.

As was suggested in Ch. 4, the representation language provides a basis for computing domain-independent heuristic functions for the planning tasks. The rest of this chapter presents such heuristics for the case of Deterministic Decision Processes, MDPs and POMDPs. For MDPs only the stochastic version is considered since the heuristic for the non-deterministic case can be obtained by straightforward modifications. In the POMDP case, however, a more detailed analysis is necessary so both cases are treated separately.

8.1 Heuristics For DDPs

Powerful heuristic functions can be obtained for planning problems expressed in the STRIPS language (Ch. 4). In this section, deterministic planning problems are assumed to be expressed as a set of *grounded* operators described by their precondition, add and delete lists, and initial and goal situation described by set of propositions. For an action a, its precondition, add and delete lists are denoted by $PREC(a)$, $ADD(a)$ and $DEL(a)$ respectively. Also, for the purpose of clarity, unit costs are assumed throughout; similar definitions and results can be obtained for the case of costs that depend on actions. However, in the most general case of costs that depend on the states and actions, the appropriate value in the equations is the minimum positive action cost.

The general idea is to use the problem description in order to assign costs for reaching *individual* atoms, from a given state, and then combine such costs to define the heuristic value for such state.

A first idea is to relax the planning problem by forgetting the delete list of each operator, or what is the same, to forget about negative interactions between operators. Then, a solution to this relaxed planning problem provides an admissible heuristic for the original problem. The difficulty however is that solving this relaxation is still `np-hard`; a fact that can be shown with a simple reduction of set covering.

Thus, instead of solving exactly such relaxation, we approximate its solution by solving the following equations.

Given a state description S, the cost $G(S,p)$ of reaching atom p from state S is defined as the solution of the equation:

$$G(S,p) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } p \in S, \\ \min \{ 1 + G(S, PREC(a)) : ADD(a) \ni p \} & \text{otherwise.} \end{cases} \tag{8.1}$$

That it, zero if p is true in S, or the minimum cost for $PREC(a)$ over all actions that assert p. Here, $G(S,A)$ denotes the cost associated to a set A of atoms, which is given by

$$G(S,A) \stackrel{\text{def}}{=} \max_{p \in A} G(S,p). \tag{8.2}$$

Having defined the cost of a set of propositions, the heuristic value for description S is defined as the cost of reaching the set of goal propositions from state S, i.e. as $h_{max}(S) \stackrel{\text{def}}{=} G(S, S_{goal})$ where S_{goal} is the description of the goal states.

Equations (8.1) and (8.2) can also be understood by observing that they correspond to Bellman's equations for a shortest-path problem in *atom space* made of the set of atoms, and unit-cost edges (p, p') where $p \in PREC(a)$ and $p' \in ADD(a)$ for some action a.

For example, in a problem with three operators $\{a_1, a_2, a_3\}$ where

$$\begin{aligned} PREC(a_1) &= \{p_1, p_2\}, & ADD(a_1) &= \{p_4\}, \\ PREC(a_2) &= \{p_1, p_4\}, & ADD(a_2) &= \{p_5\}, \\ PREC(a_3) &= \{p_3, p_5\}, & ADD(a_3) &= \{p_6\}, \end{aligned}$$

and from a state description $S = \{p_1, p_2, p_3\}$, the costs for all atoms p_1, p_2, p_3, p_4, p_5 and p_6 are 0, 0, 0, 1, 2 and 3 respectively; e.g. the cost for p_5 is given by

$$\begin{aligned} G(S, p_5) &= 1 + G(S, PREC(a_2)) \\ &= 1 + G(S, \{p_1, p4\}) \\ &= 1 + \max\{G(S, p_1), G(S, p_4)\} \\ &= 1 + \max\{0, 1 + G(S, PREC(a_1))\} \\ &= 1 + \max\{0, 1 + G(S, \{p_1, p_2\}\} \\ &= 1 + \max\{0, 1 + \max\{G(S, p_1), G(S, p_2)\}\} \\ &= 1 + \max\{0, 1 + \max\{0, 0\}\} \\ &= 2. \end{aligned}$$

Note that the delete lists play no role in the definition of the heuristic since h_{max} corresponds to a relaxed problem in which the delete lists are *ignored*. Moreover, (8.2) implies that only the most costly atom in the precondition lists account for the heuristic values, hence the shortest-path problem in atom space can be solved by a simple forward propagation of costs. The following is direct

Theorem 41 *For deterministic models defined by D1–D6 in Ch. 3, h_{max} is an admissible heuristic function that can be computed in $O(|A| + |P|)$ time and $O(|P|)$ space where A is the set of actions and P the set of propositions.*

The h_{max} heuristic was first defined in [33] and later used in the HSP planners for obtaining optimal solutions [30, 28]. However, if optimality is not crucial, the max in (8.2) can be replaced by a sum to obtain the 'additive' heuristic h_{add} which is understood as an approximation under the assumption that different atoms are independent and, hence, the cost is additively decomposable. Since this assumption does not hold in general, h_{add} is *non-admissible*.

By using somewhat related ideas, Hoffman's FF planner showed impressive results in the Second Planning Competition that granted him the first prize; see [108] for a description of FF.

As in the case of the 15-puzzle, the above ideas can be extended to consider the cost of reaching pairs of atoms (or higher order tuples) instead of single propositions. This path leads to more powerful heuristics that can be computed by replacing G with a cost function G_2 for subsets of atoms of size at most 2.

Formally, let $A(p, q)$ and $A(p|q)$ be the set of operators that respectively establish p, q and that establish p but not q. The, the equation that define G_2 are:

$$G_2(S, \{p\}) \overset{\text{def}}{=} \begin{cases} 0 & \text{if } p \in S, \\ \min\limits_{a \in A(p)} [\, 1 + G_2(S, PREC(a)) \,] & \text{otherwise.} \end{cases}$$

$$G_2(S, \{p, q\}) \overset{\text{def}}{=} \begin{cases} 0 & \text{if } p, q \in S, \\ \min\{ \min\limits_{a \in A(p,q)} [\, 1 + G_2(S, PREC(a)) \,], & \\ \qquad \min\limits_{a \in A(p|q)} [\, 1 + G_2(S, PREC(a) \cup \{q\}) \,], & \\ \qquad \min\limits_{a \in A(q|p)} [\, 1 + G_2(S, PREC(a) \cup \{p\}) \,] \,\} & \text{otherwise.} \end{cases}$$

The first equation for singletons is identical to (8.1). The second equation, on the other hand, needs to consider all possible ways of achieving the pair $\{p, q\}$: either through the operators that establish p and q, or first through the operators that establish p but not q, or through the operators that establish q but not p.

The cost of a set of atoms is defined analogously as before by

$$G_2(S, A) \overset{\text{def}}{=} \begin{cases} G_2(S, A) & \text{if } |A| < 2, \\ \max\{\, G_2(S, B) : B \subseteq A, |B| = 2 \,\} & \text{otherwise.} \end{cases}$$

The G_2 costs are used to define the h^2_{max} heuristic as $h^2_{max}(S) \overset{\text{def}}{=} G_2(S, S_{goal})$. This heuristic was defined by Haslum and Geffner in [102] where, in addition, general costs G_m of achieving m-sets of atoms are given, and thus used to define higher-order heuristics h^m_{max}. These heuristics are not only admissible but monotonic as given by the following result.

Theorem 42 *For deterministic models defined by D1–D6, the h^m_{max} heuristic is admissible and monotonic for every $m \geq 1$.*

Proof: It is sufficient to show monotonicity since admissibility is a consequence of it [155]. Observe that, by definition, $G_m(S, A) \leq G_m(S', A)$ whenever $S \supseteq S'$ for all subsets A with $|A| \leq m$. Define the set

$$ADD(S) \overset{\text{def}}{=} \bigcup \{ADD(a) : PREC(a) \subseteq S\}.$$

Fix a subset A of size at most m. If we can show that $G_m(S, A) = 1 + G_m(S \cup ADD(S), A)$ whenever $A \not\subseteq S$, then the result will follow since, for a with $PREC(a) \subseteq S$,

$$\begin{aligned} G_m(S, A) &= 1 + G_m(S \cup ADD(S), A) \\ &\leq 1 + G_m(S \cup ADD(a), A) \\ &\leq 1 + G_m(S \cup ADD(a) \setminus DEL(a), A) \\ &\leq 1 + G_m(effect(S, a), A). \end{aligned}$$

That is, $G_m(S, A) \leq 1 + G_m(S', A)$ for all successor state S' of S. Taking maximum in each side, we obtain

$$\max\{G_m(S, A) : A \subseteq S_{goal}, |A| \leq m\} \leq 1 + \max\{G_m(S', A) : A \subseteq S_{goal}, |A| \leq m\}$$

which is the definition of monotonicity; i.e. $h(S) \leq 1 + h(S')$ for all successor state S' of S.

The fact $G_m(S, A) = 1 + G_m(S \cup ADD(S), A)$ follows by a simple induction on the value $G_m(S, A)$. □

The above heuristics are only applicable to the original STRIPS formulation where operators are given in terms of the precondition, add and delete lists. The same ideas apply to the extension of STRIPS known as ADL (Ch. 4) in which operators may contain arbitrary logical preconditions and conditional effects. In such cases, and only for the purpose of computing the heuristic, each operator a with n conditional effects '`when` φ_j `do` α_j `end-do`' where α_j is a sequence of simple effects[1] is split into 2^n operators whose preconditions are the conjunction of $PREC(a)$ with the 2^n possible valuations for the sequence of φ_j, and the effects are the effects of a plus the ones that correspond to the positive valuations of the φ_j's. The heuristic is then defined as in the STRIPS case using the new set of operators.

The case for the f-STRIPS language, even without ramifications or defined predicates, is much more complex and remains as an open problem. A proposal with good experimental results for a restricted class of problems appears in [106] in the context of planning under resources. More about this will be said in Ch. 12.

8.2 Heuristics For MDPs

For the case of MDP models given by S1–S7 (Ch. 3), an admissible heuristic function can be obtained by the solving the problem associated with the equations:

$$\tilde{J}(s) = \min_{a \in A(s)} \left(g(s, a) + \alpha \min\{\, \tilde{J}(s') : p(s, a, s') > 0 \,\} \right) \tag{8.3}$$

[1]The more general case of nested conditional effects can be reduced to this case by a simple preprocessing.

where α is the discount factor that usually is 1. Since the second minimization can be pulled out in front of the first, the above equation can be understood as a relaxation in which for every operator only its more favorable effect towards reaching the goal is considered; i.e. a kind of 'optimistic' version of the problem.

The resulting heuristic, called the h_{min} heuristic, is defined using the solution $\tilde{J}^*$ of (8.3) as $h_{min}(s) = \tilde{J}^*(s)$. This heuristic has proved to be very effective in a number of problems; e.g. in the racetrack domain as is seen in the experiments of Ch. 6.

In the case of stochastic shortest-path problems, the heuristic values $h_{min}(s)$ can be computed *on demand* using any deterministic search algorithm like A*, IDA*, or a Labeled version of the LRTA* algorithm that corresponds to the deterministic version of LRTDP. The later is the preferred version since all partial computations can be reused when computing other estimates; see Ch. 6 for the definition of the LRTDP algorithm and its implementation details.

The monotonicity of the Bellman operator T (defined in (3.4)) can be used to obtain more powerful heuristics h^k_{min} by k applications of T as

$$h^k_{min}(s) \overset{\text{def}}{=} (T^k \tilde{J}^*)(s)$$

for $k \geq 0$. Obviously, h_{min} is a special case of this heuristic since $h_{min} = h^0_{min}$. Some properties of this family of heuristics are given by

Theorem 43 *For a* MDP *problem given by S1–S7 and assumption A0, the h^k_{min} heuristic is an admissible and monotonic function for all $k \geq 0$, and $h^k_{min} \leq h^{k+1}_{min}$. By admissible and monotonic is meant $h^k_{min} \leq J^*$ and $h^k_{min} \leq T h^k_{min}$ respectively.*

Proof: The solution $\tilde{J}^*$ of (8.3) can be obtained using value iteration. Denote with $\tilde{T}$ the Bellman operator for such a problem. Then,

$$\begin{aligned} \tilde{T}\tilde{J}(s) &= \min_{a \in A(s)} \left(g(s,a) + \alpha \min \{ \tilde{J}(s') : p(s,a,s') > 0 \} \right) \\ &\leq \min_{a \in A(s)} \left(g(s,a) + \alpha \sum_{s' \in S} p(s,a,s') \tilde{J}(s') \right) \\ &= (T\tilde{J})(s) \end{aligned}$$

and using induction it follows that $\tilde{J}^* \leq J^*$ so h_{min} is admissible. The monotonicity is established with

$$\begin{aligned} (T\tilde{J}^*)(s) &= \min_{a \in A(s)} \left(g(s,a) + \alpha \sum_{s' \in S} p(s,a,s') \tilde{J}^*(s') \right) \\ &\geq \min_{a \in A(s)} \left(g(s,a) + \alpha \min \{ \tilde{J}^*(s') : p(s,a,s') > 0 \} \right) \\ &= (\tilde{T}\tilde{J}^*)(s) \\ &= \tilde{J}^*(s) \end{aligned}$$

since $\tilde{J}^*$ is the fixed point of $\tilde{T}$. For $k > 0$, apply T^k and use its monotonicity. □

By this time, the reader might wonder what is the relation between the deterministic case and the MDP case, and in fact, what role the representation language plays for computing heuristics in the MDP case. There are at least two possible answers for these questions. First, the heuristic search method used to solve (8.3) can be bootstrapped by using as an initial estimate the heuristic h^m_{max} from the previous section. Since h^m_{max} is admissible for the problem defined by (8.3), the estimates returned by the search are guaranteed to be admissible.

The other possibility is to use h^m_{max} directly to define an heuristic function for the MDP. The viability of this approach is given by the following result that can be understood as a generalization of the above theorem.

Theorem 44 *Assume a* MDP *model given by S1–S7 and A0. If $\tilde{J}$ satisfies $\tilde{J} \leq \tilde{J}^*$, then $\tilde{J}$ is an admissible function for the* MDP*. In addition, if $\tilde{J}$ is monotonic for the problem defined by* (8.3)*, then it is monotonic for the* MDP.

Proof: Admissibility is direct since $\tilde{J} \leq \tilde{J}^* \leq J^*$ (see proof of Theorem 43). The monotonicity with respect to T is given by

$$\begin{aligned} (T\tilde{J})(s) &= \min_{a\in A(s)} \left(g(s,a) + \alpha \sum_{s'\in S} p(s,a,s')\tilde{J}(s') \right) \\ &\geq \min_{a\in A(s)} \left(g(s,a) + \alpha \min \{ \tilde{J}(s') : p(s,a,s') > 0 \} \right) \\ &= (\tilde{T}\tilde{J})(s) \\ &\geq \tilde{J}(s) \end{aligned}$$

by the monotonicity of $\tilde{J}$ with respect to $\tilde{T}$. □

8.3 Heuristics For POMDPs

The last and most general case is heuristics for POMDPs. In this section, three methods for obtaining such estimates are presented. The first one generalizes the QMDP policy for POMDPs given in [134] and the EVEN-ODD heuristic of [10]. The second method is based on similar ideas as the h^m_{max} heuristics and can be understood as a pattern database method for obtaining heuristics, and the third is a general version of Theorem 44 for belief space.

The QMDP heuristic is obtained by considering a relaxation in which the problem becomes *completely observable* after the first action. That is, if x is a given belief state, the relaxation assumes that after taking one action in x, the true state of the system is revealed through a non-ambiguous observation. Solving this relaxation is equivalent to solving the underlying MDP problem (i.e. the problem that results from assuming A0) and using its solution $\tilde{J}^*$ to define:

$$h_{mdp}(x) \overset{\text{def}}{=} \sum_{s\in S} x(s)\tilde{J}^*(s). \tag{8.4}$$

As before, the Bellman operator T (3.21) for the POMDP can be used to define the family h^k_{mdp} of heuristics, from which h_{mdp} is a special case, as

$$h^k_{mdp}(x) \overset{\text{def}}{=} (T^k h_{mdp})(x) \tag{8.5}$$

for belief state x and any $k \geq 0$. The h_{mdp} heuristic proved to be very powerful for a number of POMDP problems in [26]. Yet, as can be seen in (8.4), the h_{mdp} heuristic *does not* consider the sensing aspects of the actions and so it can be misleading in problems involving knowledge gathering actions. This problem was identified in [10] in which the authors proposed a more powerful heuristic, called EVEN-ODD, that results by doing one-step lookahead in belief space and then using (8.4). It follows that EVEN-ODD is just the h^1_{mdp} heuristic, and that the case for general k is just a k-step lookahead using h_{mdp}. All these heuristics are admissible, monotonic and increasing dominating as follows from

Theorem 45 *Consider a POMDP problem given by SB1–SB7 (Ch. 3), its Bellman operator T, and an integer $k \geq 0$. If J is an admissible (and monotonic) estimate then T^kJ is an admissible (and monotonic) estimate. If J is a monotonic estimate, then J is admissible and $T^kJ \leq T^{k+1}J$.*

Proof: J admissible means $J \leq J^*$ where J^* is the optimal value function (fixed point of T). The monotonicity of T implies $TJ \leq TJ^* = J^*$, and after k applications $T^kJ \leq J^*$. If J is monotonic, then $J \leq TJ$ and after $k+1$ applications of T we obtain

$$J \leq TJ \leq T^2J \leq \cdots \leq T^kJ \leq T^{k+1}J \to J^*.$$

Thus, if J is a monotonic estimate, then $J \leq J^*$. □
The case for the h_{mdp} heuristic is given by

Corollary 46 *The heuristic h^k_{mdp} is admissible and monotonic and $h^k_{mdp} \leq h^{k+1}_{mdp}$.*

Proof: First note that if h_{mdp} is monotonic then $Th_{mdp} \leq T^k h_{mdp} \to J^*$ and its admissibility follows. Therefore, it is enough to show the monotonicity of the heuristic.

$$\begin{aligned}
(Th_{mdp})(x) &= \min_{a\in A(x)} \left(g(x,a) + \alpha \sum_{o\in O(x,a)} q(x,a,o) h_{mdp}(x^o_a) \right) \\
&= \min_{a\in A(x)} \left(g(x,a) + \alpha \sum_{o\in O(x,a)} q(x,a,o) \sum_{s\in S} x^o_a(s) \tilde{J}^*(s) \right) \\
&= \min_{a\in A(x)} \left(g(x,a) + \alpha \sum_{o\in O(x,a)} \sum_{s\in S} q(s,a,o) x_a(s) \tilde{J}^*(s) \right) \\
&= \min_{a\in A(x)} \left(g(x,a) + \alpha \sum_{s\in S} x_a(s) \tilde{J}^*(s) \right) \\
&= \min_{a\in A(x)} \left(\sum_{s'\in S} x(s') g(s',a) + \alpha \sum_{s\in S} \sum_{s'\in S} x(s') p(s',a,s) \tilde{J}^*(s) \right) \\
&= \min_{a\in A(x)} \sum_{s'\in S} x(s') \left(g(s',a) + \alpha \sum_{s\in S} p(s',a,s) \tilde{J}^*(s) \right) \\
&\geq \sum_{s'\in S} x(s') \min_{a\in A(s')} \left(g(s',a) + \alpha \sum_{s\in S} p(s',a,s) \tilde{J}^*(s) \right) \\
&= \sum_{s'\in S} x(s') \tilde{J}^*(s') \\
&= h_{mdp}(x)
\end{aligned}$$

where the inequality follows since $A(x) \subseteq A(s')$ when $x(s') > 0$ and the second to last equality since $\tilde{J}^*$ is the fixed point of the underlying MDP. □

A second class of heuristic functions is obtained by studying the problem in non-deterministic belief space generated by the operator

$$(\overline{T}J)(A) \overset{\text{def}}{=} \min_{a\in A(A)} \left(\min_{s\in A} g(s,a) + \alpha \min_{o\in O(A,a)} J(A^o_a) \right) \tag{8.6}$$

where A is a set of states as in the non-deterministic version of POMDPs. The same arguments for the proof of the non-deterministic version of Theorem 12 can be used to show that (8.6) has a fixed point solution, which will be denoted as $\bar{J}^*$. Similarly, the same arguments show that the value iteration algorithm converges to this solution.

If for belief state x, we define support$(x) \overset{\text{def}}{=} \{s : x(s) > 0\}$, then our goal is to show that the heuristic function h_{sup} defined

$$h_{sup}(x) \overset{\text{def}}{=} \bar{J}^*(\text{support}(x)) \tag{8.7}$$

is an admissible and monotonic heuristic function.

Theorem 47 *For a* POMDP *model given by SB1–SB7 (Ch. 3) and belief state x, $\bar{J}^*(\text{support}(x)) \leq J^*(x)$.*

Proof: Fix a belief state x and let $A = \text{support}(x)$. Since the value iteration algorithm converges for the problem (8.6), it is enough to show that $(\overline{T}^k \bar{J})(A) \leq J^*(x)$ for all $k \geq 0$ where $\bar{J}$ is a constant function such that $\bar{J}(A) \leq$

$J^*(x)$; e.g. the identically zero value function in the case of positive costs. The proof is by induction on k showing $(\overline{T}^k \bar{J})(A) \leq J^*(x)$. The base case is trivial by the choice of $\bar{J}$. For the inductive step

$$\begin{aligned}
(\overline{T}^{k+1} J)(A) &= \min_{a\in A(A)} \left(\min_{s\in A} g(s,a) + \alpha \min_{o\in O(A,a)} (\overline{T}^k \bar{J})(A_a^o) \right) \\
&\leq \min_{a\in A(A)} \left(\min_{s\in A} g(s,a) + \alpha \min_{o\in O(A,a)} J^*(x_a^o) \right) \\
&= \min_{a\in A(x)} \left(\min_{s\in A} g(s,a) + \alpha \min_{o\in O(x,a)} J^*(x_a^o) \right) \\
&\leq \min_{a\in A(x)} \left(\min_{s\in A} g(s,a) + \alpha \sum_{o\in O(x,a)} q(x,a,o) J^*(x_a^o) \right) \\
&\leq \min_{a\in A(x)} \left(g(x,a) + \alpha \sum_{o\in O(x,a)} q(x,a,o) J^*(x_a^o) \right) \\
&= (TJ^*)(x) \\
&= J^*(x),
\end{aligned}$$

where the first inequality follows by the fact $A_a^o = \text{support}(x_a^o)$ and the inductive hypothesis. □

Theorem 48 *For a* POMDP *model given by SB1–SB7 (Ch. 3), the heuristic h_{sup} is admissible and monotonic.*

Proof: The admissibility follows from above. For monotonicity, fix a belief state x and denote its support with A. Then,

$$\begin{aligned}
(Th_{sup})(x) &= \min_{a\in A(x)} \left(g(x,a) + \alpha \sum_{o\in O(x,a)} q(x,a,o)\, h_{sup}(x_a^o) \right) \\
&= \min_{a\in A(A)} \left(g(x,a) + \alpha \sum_{o\in O(A,a)} q(x,a,o)\, \bar{J}^*(A_a^o) \right) \\
&\geq \min_{a\in A(A)} \left(\min_{s\in A} g(s,a) + \alpha \min_{o\in O(A,a)} \bar{J}^*(A_a^o) \right) \\
&= h_{sup}(x).
\end{aligned}$$

□

In the non-deterministic case, a better family of heuristics can be obtained by considering the following problem on belief states with size at most m:

$$\begin{aligned}
&(\hat{T}J)(A) \overset{\text{def}}{=} \\
&\min_{a\in A(A)} \left(\max_{s\in A} g(s,a) \;+\; \alpha \max_{o\in O(A,a)} \max\{\, J(A_a^o \cap B) : B \cap A_a^o \neq \emptyset,\, |B| = m \,\} \right)
\end{aligned} \tag{8.8}$$

Denote the fixed point solution of this equation with $\hat{J}_m^*$ and, remembering that in the non-deterministic case belief states are just subsets of states, define the heuristic

$$h_{sup}^m(A) \overset{\text{def}}{=} \max\{\hat{J}^*(A \cap B) : B \subseteq S, |B| = m\}. \tag{8.9}$$

Observe that this heuristic is defined by the value of $\hat{J}^*$ over all subsets of size at most m, so it looks like a pattern database for non-deterministic problems in belief space. The admissibility and monotonicity of the heuristics are given in the following theorem whose proof is essentially the same as in the previous cases.

	transition model				
	state space			belief space	
	det.	non-det.	stochastic	non-det.	stochastic
kernel	h^m_{max}	h_{min}	h_{min}	h_{mdp}	h_{mdp}
				h^m_{sup}	h_{sup}

Table 8.1: Kernel heuristic functions for different models. These are later improved using the corresponding Bellman operators.

Theorem 49 *For a non-deterministic* POMDP *model, $\hat{J}^* \leq J^*$, and the heuristic h^m_{sup} is admissible and monotonic.*

By Theorem 45, the heuristics in (8.7) and (8.9) can be 'boosted' with the k-fold composition of T to obtain a whole new family of admissible and monotonic heuristics for POMDPs.

The last method is given by a generalization of Theorem 44 for the case of POMDPs. The proof, similar to the ones for the h_{mdp} and h^m_{sup}, is left to the reader.

Theorem 50 *Assume a* POMDP *model given by SB1–SB7. If $\tilde{J}$ is an admissible (and monotonic) value function for the underlying* MDP*, then the heuristic that results from replacing $\tilde{J}^*$ with $\tilde{J}$ in* (8.4) *is an admissible (and monotonic) heuristic. If $\bar{J}$ is an admissible (and monotonic) value function for the problem* (8.6)*, then the heuristic that results from replacing $\bar{J}^*$ with $\bar{J}$ in* (8.7) *is an admissible (and monotonic) heuristic, and similarly for the non-deterministic case.*

Thus, for example, the max heuristic for deterministic problems can be used to induce an admissible and monotonic heuristic for POMDP problems.

8.4 Summary and Notes

Heuristic functions for the different transition models are presented. The heuristics are further divided into classes that are generated by a *kernel* heuristic function and the corresponding Bellman operator for each case: deterministic, non-deterministic or stochastic either in state space or belief space. The kernel heuristics are shown in Table 8.1. All these heuristics are shown to be admissible and monotonic, and can be considered as domain-independent heuristics for the corresponding models since no other information besides the description of the problem is used to compute them.

The heuristics for the deterministic case were first defined in [33, 102] and they have a close relation to McDermott's regression estimators [143]. That these heuristics are monotonic is easy to show but, up to my knowledge, no reference to such fact appears in the literature.

The heuristic function inside the FF planner is computed using the planning graph, as found in the work of Graphplan [21], that results from the relaxation of forgetting about the delete lists. The resulting heuristic, which is non-admissible, is related to the h_{add} heuristic found in the HSP planners.

The h_{min} heuristic is a general heuristic for non-deterministic and stochastic MDP problems. The first general formulation of the heuristic appears in [32] but a restricted version for the racetrack problem appears in [99]. The generated family $T^k h_{min}$ is new and generalizes previous work on lookahead heuristics as the one found in [121].

The h_{mdp} heuristic is inspired by the QMDP policies of [134]. It has been used to solve large problems in [26]. The one-step lookahead version Th_{mdp} is exactly the EVEN-ODD heuristic in [10]. The general definition for k-step lookahead family is new.

The h_{sup} heuristics for non-deterministic and stochastic POMDPs are new. The stochastic version can be thought as an instance of the h_{min} heuristic. In the non-deterministic case, the family h^m_{sup} corresponds to an instance of the ideas found in h^m_{max} for deterministic problems. Such heuristics can be thought as a pattern database since it involves solving the problem for belief states with at most m states.

Finally, a method for computing the h_{mdp} heuristic for a special case of non-deterministic problems is given in [35]. The method is based on an IDA* algorithm with transposition tables.

Chapter 9

Experimental Results

Most of the methods and techniques presented in this dissertation have been implemented into a planning system called General Planning Tool or GPT. The input to GPT is a problem description in a slight modification of the description language presented in Ch. 4 that makes it very similar to the PDDL language for classical planning [145]. The main difference is in that PDDL (and in GPT's language) the domain and problem instance descriptions are in separated syntactic units which is convenient when multiple problem instances of a given domain are considered.

The GPT system works by parsing the description language and generating an *implicit* representation of the corresponding mathematical model into a $C++$ program. That program is compiled and linked with some standard libraries and then dynamically loaded into memory. The, GPT uses domain-independent implementations of the solving algorithms to find an optimal solution for the problem instance. Among other things, GPT also offers functionality for policy evaluation, model debugging at the level of the underlying MDP and belief-MDP, generation of a policy description either in tabular or graphical form, etc.

At the current date, the author is not aware of any other automated tool for computing *optimal* policies for such diverse problem types as the ones supported by GPT; i.e. deterministic, conformant and contingent planning problems, both either non-deterministic and stochastic. This fact makes it difficult to assess the benefits and limitations of the approach.

As pointed out in Ch. 7, there are some systems (based on Sondik's representation) that find optimal policies for general POMDP problems, yet the input representation language is an explicit one comprised of transition matrices that does not support action preconditions. Thus, most of the problems over which GPT has been tested are not even directly expressible in such languages.

Perhaps the closest system to GPT are the two MBP planners of [13, 14]. The first is an optimal planner for conformant planning problems, while the second is a *non-optimal* planner for contingent planning problems. In both cases, the MBP planners only focus in non-deterministic problems with unitary action costs and thus, in the conformant case, the planner only minimizes the solution length.

The GPT system has evolved over four years from a first version based on the RTDP algorithm, and recently it has been extended with the LRTDP algorithm. During this time, GPT has been tested on a large number of problems ranging from identification and localization problems, medical diagnosis, navigation problems, synthesis of circuits and decision trees, power restoration, etc.

Instead of trying GPT over all such domains, we have selected few domains with different characteristic with respect to solution depth, branching factor and number of states. GPT is evaluated over different instances in each case in order to identify the strengths and weaknesses of the approach, and to see how far the proposed methodology can be pushed with the current search algorithms and heuristic functions. As will be seen, starting from sound and basic principles, GPT is able to optimally solve a number of non-trivial planning tasks, yet the current heuristics and algorithms don't scale well in larger instances for different reasons. However, as the experiments show (and the author believes), the results demonstrate that GPT is on the 'right track' towards achieving a reasonable GPS for sequential decision making involving uncertainty and partial information.

This chapter is organized as follows. In the next section, GPT is evaluated over different instances of Levesque's

	problem instance					
	om-3	om-4	om-5	om-6	om-7	om-8
time	7.11	42.45	247.24	1,246.16	5,355.79	23,272.16
MDP states	300	675	1,323	2,352	3,888	6,075
visited beliefs	1,189	4,804	15,785	44,843	113,798	263,659
optimal cost	23.00	31.00	39.00	47.00	55.00	63.00
heuristic	10.46	14.17	17.91	21.68	25.46	29.25

Table 9.1: Solution times in seconds, number of states, number of belief states, and optimal cost and heuristic value for the initial belief for different instances of the omelette problem with $n = 3, \ldots, 8$.

omelette problem given in Ch. 4. In the second section, GPT is tested over several instances of the game of Mastermind. In the third section, a conformant planning problem of synthesis of sorting networks is considered. Finally, in the fourth section, GPT is tested on the power supply restoration domain that was proposed as a challenging benchmark in [194]. At the end, some conclusions and notes are given.

9.1 The Omelette Problem

The omelette problem consists of an agent that has a large supply of eggs and whose goal is to get n good eggs and no rotten ones into one of two bowls. The eggs can be either good or rotten, and the agent can perform tasks as breaking an egg into a bowl, pouring the content of a bowl into another, and cleaning a bowl. The agent can also sense if a bowl contains a rotten egg by smelling it; an action that returns a boolean value denoting whether the bowl contains at least one rotten egg.

A formal description of an instance of this problem for $n = 3$ appears in Fig. 4.3 in Ch. 4.

GPT was tested on several instances of this problem for an increasing number of eggs from 3 to 8. As n increases, the number of states in the underlying MDP as well as the number of relevant belief states increases exponentially. Table 9.1 shows the running time in seconds, number of states in the underlying MDP, number of visited belief states, the optimal cost and the heuristic value for the initial belief for the LRTDP algorithm equipped with the h_{QMDP} heuristic function over different instances.

As can be seen, GPT is able to solve small instances but the combinatorial explosion and the lack of a more informative heuristic precludes GPT from finding optimal solutions for larger instances. The main difficulty is the depth of the optimal solutions and the big gap between the optimal costs and the heuristic values as seen in the bottom rows of Table 9.1.

Interestingly, the optimal solution for this problem is known for general n. Fig. 9.1 shows a graphical representation of the optimal policy for the case $n = 3$. This representation is a labeled graph where the nodes correspond to belief states, and the edges to possible transitions between belief states. The graph contains labels for both nodes and edges. Each node is labeled with the optimal action for the corresponding belief state, while each edge is labeled with an observation outcome associated with the application of the optimal action in the belief state corresponding to the tail of the edge. For example, in Fig. 9.1, the out-edges for each `inspect` node are labeled as `good` or `bad` since the `inspect` action has as possible observations such values. The other edges has no labels since there are no observations for the corresponding actions.

Remember that the goal in this instance is to obtain three good eggs and no rotten ones in one of two bowls. Observe that the policy in Fig. 9.1 always achieves the three good eggs in the small bowl. At first sight, it seems that a better policy could be obtained by delaying the decision of choosing the final bowl, and thus saving a `pour` operation in some trajectories. Yet, the policy in Fig. 9.1 incurs exactly in two `pour` operations in each trajectory of the system, so the former argument does not hold. Indeed, the policy of Fig. 9.1 is optimal for the case $n = 3$. Similar optimal policies for bigger n can be obtained using more ‘loops’.

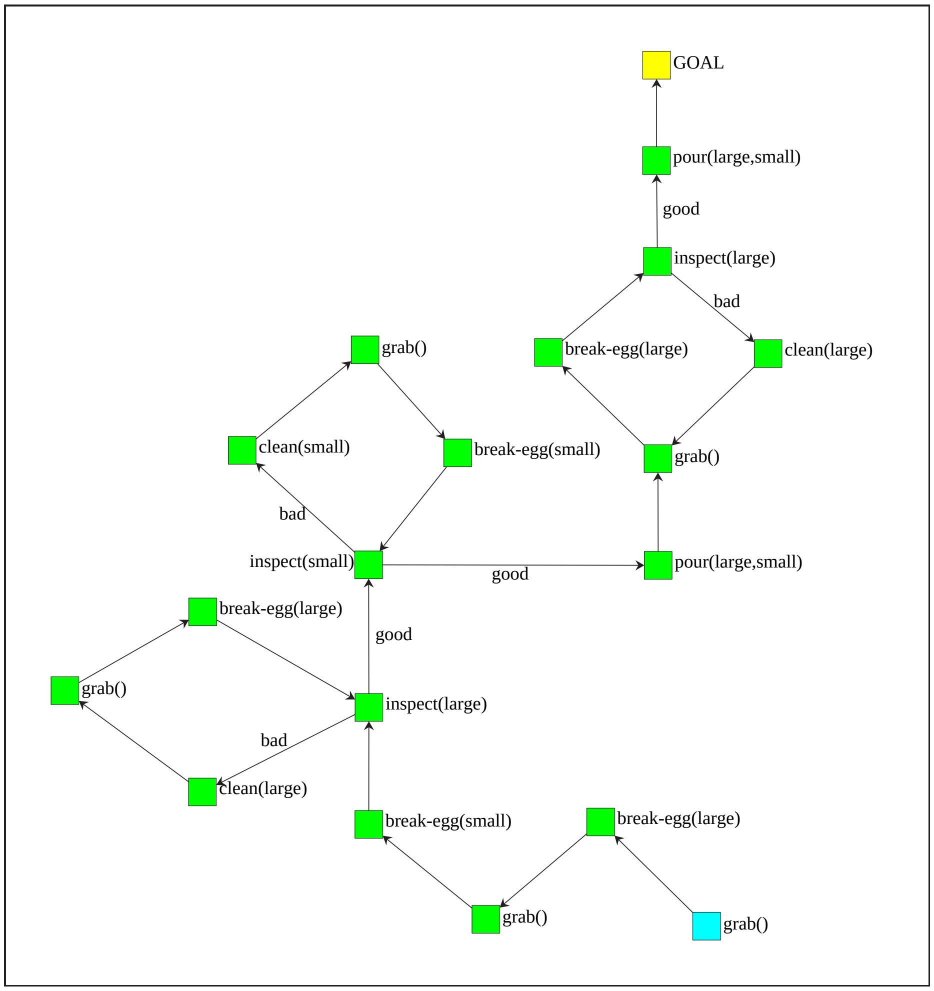


Figure 9.1: Optimal policy for the omelette problem with three eggs (`om-3`).

9.2 The Game of Mastermind

In this second experiment, GPT is confronted with several instances of the game of Mastermind.

Remember that the Mastermind game consists of two players in which the first player chooses a secret code made of 4 colored pegs each one of 6 possible colors, and that the task of the second player is to discover the code by making guess. Each guess made by the second player is 'marked' by the first player with smaller black and white pegs that tells how many pegs in the guess are of the right color and in the right position (black marks) and how many pegs are of the right color but in the wrong position (white marks); a more precise description of the game together with some examples can be found in the introduction.

We have tested GPT on different problem instances of Mastermind with a number of pegs that range from 3 to 6, and a number of colors from 2 to 6.

Table 9.2 show the experimental results obtained. The table contains the solution time in seconds, the worst-case optimal cost, the action branching factor (which is equal to the number of states in the underlying MDP), the number of observations and the number of visited beliefs.

As can be seen, GPT is able to solve some small instances but the problem quickly becomes very hard. It is important to note that the heuristic function in this problem is identically zero. Other more informative heuristic, like h_{sup} from in Ch. 8, could work in this problem. As of today, such new heuristic functions have not been implemented

	3 pegs				
	2 colors	3 colors	4 colors	5 colors	6 colors
time	0.23	1.73	248.89	2,639.70	—
cost	2	2	3	3	—
branch.	8	27	64	125	216
obs.	8	11	11	11	—
beliefs	17	324	9,378	70,804	—

	4 pegs			5 pegs		6 pegs
	2 colors	3 colors	4 colors	2 colors	3 colors	2 colors
time	0.98	1,128.36	—	17.41	—	1,381.43
cost	3	3	—	4	—	5
branch.	16	81	256	32	243	64
obs.	10	16	—	12	—	14
beliefs	123	13,449	—	1,293	—	20,009

Table 9.2: Results for the game of Mastermind for instances with different numbers of pegs and colors. The table includes time in seconds, optimal worst-case cost, branching factor (= MDP states), number of observations and number of visited beliefs. A dash (—) means that GPT was not able to solve the problem in 20 hours.

in GPT.

Although the heuristic is non-informative, the main difficulty in this problem lies in the exponential blow of the branching factor. This fact makes each belief expansion, and hence each LRTDP trial, very slow. Thus, although the number of visited belief states is not big and the optimal solution is shallow, the number of trials needed for convergence is still significant and so convergence is very slow; e.g., for 3 pegs and 5 colors GPT needs $1,630$ trials each taking 1.61 seconds on average for a total time of $2,638.68$ seconds, while for 3 pegs and 4 colors 680 trials are needed each taking 0.36 seconds for a total time of 248.41 seconds.

Not much is known about the game of Mastermind. In the earliest reference I am aware of, Knuth [118] shows that 4 is the minimum number of guesses needed to discover the secret code for a game with 4 pegs and 6 colors. Knuth's strategy is a greedy strategy that always minimizes the size of the biggest set of remaining codes compatible with the possible answer for the guess. Until now, there is no proof (nor a counterexample) showing that such greedy strategy always produces an optimal strategy. This fact, however, seems to be very unlikely.

Chvátal [50] derives upper and lower bounds for the minimum number of guesses needed to unveil the secret code; up to my knowledge these are the tightest bounds known for Mastermind. Following Chvátal, let $f(n,k)$ denote the minimum number of guesses to discover the secret code in a game with n pegs and k colors. Then, $f(n,k) \geq n \log k / \log \binom{n+2}{2}$ since there are at most $\binom{n+2}{2}$ answers to each guess, and all the $f(n,k)$ guesses have to distinguish between every two of the k^n possible secret codes. The bound on the number of answer comes from considering all pairs $\{(i,j) : 0 \leq i < j \leq n\}$ plus all pairs of the form (i,i) that are denoted with the subset $\{i,\star\}$ with the symbol $\star$; i.e., all guesses can be denoted with subsets of size 2 from $\{0,1,\ldots,n,\star\}$.

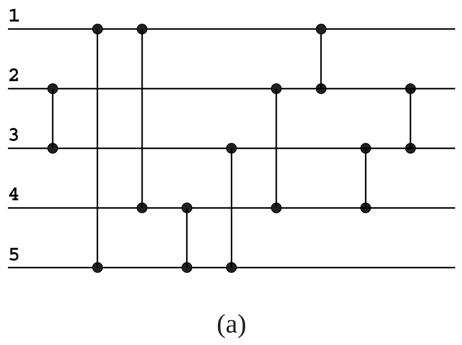
The upper bound found by Chvátal is:

Theorem 51 (Chvátal [50]) *If $n \leq k \leq n^2$, then $f(n,k) \leq 2n \log_2 k + 4n$.*

9.3 Sorting Networks

A sorting network refers to a sorting algorithm in which comparisons and swaps are merged into a single operation that takes two entries i and j and swaps them if and only if they are not ordered. Given n entries with different integers, the goal is to find a minimum-length sequence of operations that sorts the entries in ascending order no matter what are initial integers in the entries. See [117, 53] for some results about sorting networks.

Fig. 9.2(a) shows a graphical representation of a sorting network for 5 elements. The entries are represented as numbered horizontal lines while each operation as a small vertical lines joining two horizontal lines. Each operation



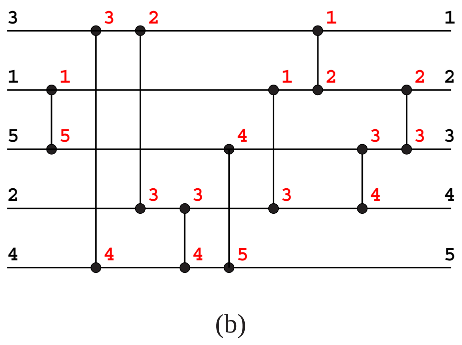


Figure 9.2: Optimal sorting network for 5 entries: (a) network layout, (b) network operation on input $(3, 1, 5, 2, 4)$.

SORTNET(n)	states	length	$h = h_{max}$	$h = 0$
SORTNET(2)	2	1	0.059	0.053
SORTNET(3)	6	3	0.061	0.061
SORTNET(4)	24	5	0.060	0.065
SORTNET(5)	120	9	0.688	0.653
SORTNET(6)	720	12	119.544	164.482
SORTNET(7)	5040		—	—

Table 9.3: A dash (—) means that GPT ran out of memory. All times are in seconds.

then swaps the content of both lines (entries) so that the minimum is moved up and the maximum down. Thus, for example, if the input lines contains the integers $(3, 1, 5, 2, 4)$, then the contents of each line are shown in Fig. 9.2(b) as the operations are applied. The network shown in Fig. 9.2(a) is optimal for 5 elements.

The problem of finding a sorting network can be cast as a conformant planning problem in which the states of the system are the possible permutations of n integers. Initially, all $n!$ permutation are possible, yet after the application of an operation only those permutations compatible with the operation remain. The goal is then to find a sequence of operations that achieve the permutation where all integers are sorted in ascending order.

The optimal (minimum-length) sorting network is only known for small values of n up to $n \leq 8$ according to [117].

We have tested GPT over this domain with the h^1_{sup} heuristic of Ch. 8 and $h = 0$ for different values of n. The results are shown in Table 9.3. As can be seen, the heuristic does not help much in this problem, still GPT finds optimal solutions in a couple of minutes for n's up to 6. It is worth to remark that GPT implements conformant planning as an A* search in belief space so the memory problem could be removed by using IDA*. However, an IDA* implementation would have to deal with difficult problems as the removal of redundant paths in the search tree. Nonetheless, we think that a domain-specific IDA* implementation could find new optimal sorting networks for small n's like 8 and 9.

9.4 Power Supply Restoration

The Power Supply Restoration (PSR) problem consists of a faulty power distribution network that has to be reconfigured in order to minimize the costs associated with unsupplied lines. The control actions are either to open or close switches so to control the flow of electricity in the network. The network is equipped with actuators that implement the control actions, and sensors that return information about the status of the network. Due to sensor and actuator uncertainty, the location of the faulty areas and the network configuration are uncertain, which leads to a tradeoff between acting to resupply lines and acting (intrusively) to reduce uncertainty. [15] reports the first successful attack of the MBP planner on the PSR domain: MBP was able to solve non-trivial instances of the problem of generating a contingent plan guaranteed to achieve a given supply restoration goal. By contrast, the objective here is to have GPT address the full scope of the PSR benchmark, for which there is no specified goal but rather the request to minimize breakdown costs. This amounts to generating *optimal* policies for a contingent planning problem, which is typically much harder.

In addition to reporting an interesting case study, this study makes a number of contributions towards the use of PSR as a benchmark for planning under uncertainty. For instance, a concise, formal and problem-independent encoding matching the informal description of PSR in [194], is given here which can be used by others as a challenging benchmark and useful in future planning competitions.

The section starts with an informal overview of PSR and then a formal encoding of the domain in the language of Ch. 4. Then, experimental results under different optimization criteria and network topologies are presented together with a brief final discussion. This section is based on [35].

9.4.1 The PSR Benchmark

In [194], the authors cast the problem of supply restoration in power distribution systems as a benchmark for planning under uncertainty. They give an informal description of the PSR benchmark (which is now summarized), and issue the challenge of modeling and solving PSR with general purpose planning tools.

Network Topology

For the purpose of the benchmark, a power distribution system (see Fig. 9.3) consists of electric lines and devices of two types: circuit-breakers (large squares in the figure) and switches (small squares). Devices are connected to at most two lines and have two possible positions: either closed or open (open devices, e.g. `SD19`, are white in the figure). Circuit-breakers are viewed as power sources.[1] When closed, they feed power into the network, and that power flows through the various lines up to the point where it is stopped by an open device. The positions of the devices are initially set so that each circuit-breaker feeds a different area of the network (the area fed by `CB4` is boxed in the figure; adjacent areas fed by different circuit-breakers are distinguished using dark and gray). The current network configuration can be modified by opening or closing devices. Closing and opening are the only available actions in PSR.

Faults and Supply Restoration

Under bad weather, lines are often affected by permanent faults. When a line is faulty, the circuit-breaker feeding it opens to prevent overloads. This leads to not just the faulty line but the entire area the breaker was feeding to be out of power. Supply restoration consists in reconfiguring the faulty network so as to minimize breakdown costs: ideally it is wanted to open and close devices in such a way as to isolate the faulty lines and resupply a maximum of the non-faulty lines on the lost areas. PSR would be relatively easy to solve if the exact locations of the faulty lines and the current network configuration were known. For instance, in case of a fault on `l20` leading `CB4` to open and the boxed area to be left without power, an adequate restoration plan, provided complete knowledge of the network state, would be to open `SD16` and `SD17` to isolate the fault, close `CB4` to resupply `l22` and `l21`, and close `SD15` to have `CB7` resupply `l19`. Unfortunately, PSR is much more complicated, because as is explained below, the sensors used to locate the faults and determine the devices' positions, as well as the actuators used to open/close devices, are unreliable.

Sensors

Each device is equipped with a fault detector and a position detector, both continuously providing action-independent sensing information. The role of the position detector is to indicate the device's current position. Position detectors have two modes: 'normal' in which the information they provide is correct, and 'out of order' in which they do not provide any information at all. The role of the fault detectors is to help locate faults. Fault detectors have one 'normal' mode and two failure modes: 'out of order' and 'liar'. In normal mode, the fault detector of a fed device indicates

[1]The name circuit-breaker comes from the literature of electric distribution networks.

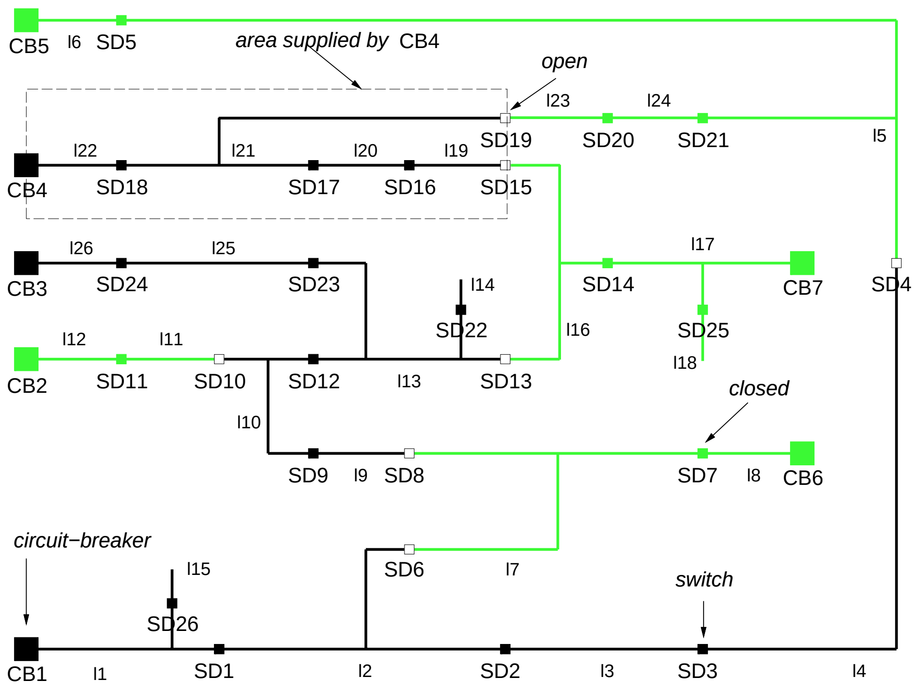


Figure 9.3: Rural Network in [194]

whether or not it is upstream of a faulty line located on the same area – upstream is to be taken in relation to the flow of current whose source is a circuit-breaker feeding the area. When not fed, a normal fault detector indicates the same information as when it was last fed. For instance, if only `l20` is faulty, only the fault detectors of `SD17` and `SD18` should indicate that they are upstream of a fault; then `CB4` will open and the information provided by the fault detectors of the devices in the lost area should remain the same until they are fed again. In 'liar' mode, fault detectors return the negation of the correct fault status, and in 'out of order' mode, they do not return any information at all.

Actuators

Each device is also equipped with an actuator whose role is to execute opening/closing actions and report on their execution status. Actuators also have a 'normal', 'out of order' and 'liar' modes. In normal mode, the actuator of the prescribed device executes the requested action and sends a positive notification. In 'out of order' mode, the actuator fails to execute the action (the position remains unchanged) and sends a negative notification. In 'liar' mode, it also fails to execute the action but still sends a positive notification. In this domain, all sensor/actuator modes are assumed to be constant across a supply restoration episode.

Minimizing Cost under Uncertainty

Under sensor and actuator uncertainty, PSR takes on another dimension. Many fault location and network configuration hypotheses are consistent with the observations, and each of them corresponds to a hypothesis about the behavior modes (normal, liar) of the sensors and actuators. Since observations can only change when a device is operated, there is no non-intrusive way of gathering information to eliminate hypotheses. Often, decisive information comes at the price of an increase in breakdown costs: the best option to determine whether a line is faulty may be to resupply it via an healthy circuit-breaker, and check whether that breaker opens, yet leading a new area to be temporarily lost! The breakdown cost for a network is defined to be proportional to the number of non-faulty lines that are not fed. In the most general case, the breakdown cost is a function of time (i.e. at each time point there is a cost associated to the current network configuration) that is integrated over the period of time from the appearance of the fault up to the end of the reconfiguration episode, and this consolidated cost is defined to be the cost for the reconfiguration. In our case, we do not integrate the cost over time but define the reconfiguration cost as the breakdown cost at the end of the reconfiguration episode.

Minimizing breakdown costs amounts to trading off the need to act to resupply lines against that of acting to reduce uncertainty. This optimization problem is extremely challenging. To the best of our knowledge, even domain-specific solvers compute suboptimal solutions, see e.g. [195].

I do not know the computational complexity of the PSR domain or some simplifications yet, in its full scope, there are questions whose answers are undecidable; see the example in Fig. 9.6 below.

9.4.2 Encodings of PSR

Figures 9.4 and 9.5 show encoding of the full PSR domain together with a problem instance with 4 faulty lines at unknown locations for the `simple` network in Fig. 9.7. The most important elements of the encoding are now described.

States

The basic types of `DEVICE` and `LINE` are needed, together with a type `SIDE` and two objects `side1` and `side2` to distinguish between the two sides of each device to which lines can connect, as well as a type `MODE` and objects `ok`, `out`, and `liar` denoting the possible fault detector and actuator modes (for a position detector mode, a simple boolean suffices).

Each state of PSR is represented as follows. The fluents `closed`, and `faulty` have the obvious readings. The functional fluents `ac_mode` and `fd_mode` return the modes of the actuator and fault detector of a device, while `pd_ok` tells whether its position detector behaves normally, and `fault_status` tells whether the device was upstream of a fault when last fed. Other fluents refer to the topology of the network: `ext` tells whether a given line is connected to a given side of a device, `breaker` tells whether a device is a circuit-breaker, and `opposite` maps `side1` to `side2` and vice versa. The initial values of functions and predicates are given in the problem instance definition. Some values may not be completely known. In the example given in Figures 9.4 and 9.5 all values are known, except that of `faulty`: there is just a constraint that the number of faulty lines equals 4. Note that propositions not mentioned at all in the initial state are taken to be false. By inspecting the domain and problem definitions, GPT is able to identify and compile away those propositions with a fixed value across the entire set of states.

Axioms

One of the difficulties of PSR is that actions have relatively complex effects: e.g. when a device is closed, a faulty line may become fed and affect devices upstream of it in some way (breakers open, and switches have their fault status changed). Being upstream is a dynamic notion which depends on the current network configuration, and so needs to be computed after each action execution. Furthermore, computing it requires an iterative or recursive traversal of the network's paths, and there is no intuitive way of doing this in the body of an ADL-style action. Consequently, to model PSR actions while keeping the encoding independent of a particular network, the predicate upstream needs to be axiomatized using a *derived* predicate. Such a predicate is one whose value is derived from the current value of other predicates and functions and cannot be directly modified by actions.

The best option to define derived predicates such as `upstream` is to use directional recursive defined predicates such as the ones described in Ch. 4. Also observe, that the `upstream` predicate is a kind of transitive closure of 'open paths' and so cannot be defined, for general networks, using first-order logic.

In the encoding, `upstream(?x,?sx,?y,?sy)` means that the current produced by some circuit-breaker flows from side `?sx` of device `?x` to side `?sy` of device `?y`. It is necessary to speak of devices' sides rather than devices in order to keep proper track of the direction of current flow. The three conjuncts respectively ensures that the current flows (1) up to side `?sx` of `?x`, (2) through `?x`, and (3) on as far as side `?sy` of `?y`.[2] Four other predicates, which

[2] In the experiments, a more efficient but longer encoding of `upstream` and other predicates is used, which is omitted for readability.

```
types     = DEVICE SIDE LINE MODE

relations = ext(LINE,DEVICE,SIDE)
            breaker(DEVICE)
            closed(DEVICE)
            faulty(LINE)
            fault_status(DEVICE)
            pd_ok(DEVICE)

functions = ac_mode :  DEVICE -> MODE
            fd_mode :  DEVICE -> MODE
            opposite :  SIDE -> SIDE

objects   = side1 side2 - SIDE
            ok out liar - MODE
            done - :boolean
            l1 l2 l3 l4 l5 l6 l7 - LINE
            cb1 cb2 cb3 sd1 sd2 sd3 sd4 sd5 sd6 sd7 - DEVICE

con(?x,?sx,?y,?sy)
    types: ?x, ?y - DEVICE, ?sx, ?sy - SIDE
  defined: (and (or ¬(?x = ?y) ¬(?sx = ?sy))
                (exists ?l - LINE (and ext(?l,?x,?sx) ext(?l,?y,?sy))))

upstream(?x,?sx,?y,?sy)
    types: ?x, ?y - DEVICE, ?sx, ?sy - SIDE
  defined: (and (or breaker(?x)
                    (exists ?z - DEVICE
                      (exists ?sz - SIDE
                        (and con(?z,opposite(?sz),?x,?sx)
                             upstream(?z,?sz,?x,?sx)))))
                closed(?x)
                (or con(?x,opposite(?sx),?y,?sy)
                    (exists ?z - DEVICE
                      (exists ?sz - SIDE
                        (and closed(?z) con(?z,opposite(?sz),?y,?sy)
                             upstream(?x,?sx,?z,?sz))))))

affected(?x)
    types: ?x - DEVICE
  defined: (exists ?l - LINE
             (and faulty(?l)
                 (exists ?y - DEVICE
                   (exists ?sy - SIDE
                     (and ext(?l,?y,?sy)
                          (exists ?sx - SIDE upstream(?x,?sx,?y,?sy)))))))

fed(?x)
    types: ?x - DEVICE
  defined: (or (and breaker(?x) closed(?x))
               (exists ?y - DEVICE
                 (exists ?sy - SIDE
                   (exists ?sx - SIDE upstream(?y,?sy,?x,?sx)))))

fed_line(?l)
    types: ?l - LINE
  defined: (exists ?x - SWITCH
             (exists ?sx - SIDE
               (and ext(?l,?x,?sx) closed(?x) fed(?x))))
```

Figure 9.4: Encoding of PSR domain and `simple` problem: types, relational, functional and constant symbols, and defined predicates.

```
status_ramification(?x)
    types: ?x - DEVICE
    ramif: when (fed ?x) do
             fault_status(?x) := affected(?x)
           end-do
open_ramification(?x)
    types: ?x - DEVICE
    ramif: when (and breaker(?x) affected(?x) do
             closed(?x) := false
           end-do
open(?x)
    types: ?x - DEVICE
   effect: when (ac_mode(?x) = ok) do closed(?x) := false end-do
      obs: (ac_mode(?x) = out),
           (list ?y - DEVICE if pd_ok(?y) closed(?y) false fi),
           (list ?y - DEVICE
             if (fd_mode(?x) = ok)
                fault_status(?x)
                if (fd_mode(?x) = liar) ¬fault_status(?x) false fi
             fi)
close(?x)
    types: ?x - DEVICE
   effect: when (ac_mode(?x) = ok) do closed(?x) := true end-do
      obs: (ac_mode(?x) = out),
           (list ?y - DEVICE if pd_ok(?y) closed(?y) false fi),
           (list ?y - DEVICE
             if (fd_mode(?x) = ok)
                fault_status(?x)
                if (fd_mode(?x) = liar) ¬fault_status(?x) false fi
             fi)
finish
   effect: done := true
     cost: (sum ?l - LINE if (or faulty(?l) fed_line(?l)) 0 5 fi)
init: done := false,
      opposite(side1) := side2, opposite(side2) := side1,
      breaker(cb1) := true, breaker(cb2) := true, breaker(cb3) := true,
      ext(l1,cb1,side2) := true, ext(l1,sd6,side1) := true,
      ext(l2,sd6,side2) := true, ext(l2,sd5,side1) := true,
      ext(l2,sd7,side2) := true, ext(l3,sd5,side2) := true,
      ext(l3,sd1,side1) := true, ext(l4,sd1,side2) := true,
      ext(l4,sd2,side2) := true, ext(l4,sd3,side2) := true,
      ext(l5,cb2,side2) := true, ext(l5,sd4,side1) := true,
      ext(l6,sd2,side1) := true, ext(l6,sd4,side2) := true,
      ext(l6,sd7,side1) := true, ext(l7,cb3,side2) := true,
      ext(l7,sd3,side1) := true,
      foreach ?x - DEVICE do
        closed(?x):= true,
        fd_mode(?x) := ok,
        pd_ok(?x) := true,
        ac_mode(?x) := ok
      end-do,
      closed(sd3) := false, closed(sd5) := false, closed(sd7) := false,
      foreach ?l - LINE do faulty(?l) := { true, false } end-do,
      prune ((sum ?l - LINE if faulty(?l) 1 0 fi) = 4)
goal: (done = true)
```

Figure 9.5: Encoding of PSR domain and `simple` problem: ramification, action schemata, initial and goal situations.

are simple abbreviations, are also defined: `con` which tells whether two devices' sides are connected via a given line, `affected` which tells whether a device is upstream of a faulty line, `fed` which tells whether a device is fed by checking that there is a side of a device upstream of one of its sides (or that the device is a closed breaker), and `fed_line` which tells whether a line is fed by checking that it is connected to a closed fed device.

If a pure PDDL encoding (without axioms) is required, it is possible to automatically compile axioms into additional context-dependent effects of existing actions or into additional actions. However, this leads to gross inefficiency, and in the worst-case, to exponentially larger domain descriptions or exponentially longer plans [196].

Actions and Observations

With the help of the derived predicates and GPT's ramification rules, PSR actions can be expressed very concisely. While the effects of an action become true at the next time step, those of a ramification rule become true immediately. These rules are useful to specify indirect action effects and domain constraints. When an action is executed, it effects are computed, then the ramification rules are applied to the resulting state in the order they appear, and only then the action's observations and cost are evaluated. Note that the ramification rules are also applied in the initial state, before any action takes place.

In the encoding of the full PSR, PSR actions have no preconditions. Their effect is simply to change the position of the device unless the actuator is abnormal. Then the ramification rules take care of setting the fault status of fed devices appropriately (if a device is affected its status is set and otherwise unset), and of opening affected breakers. The observations include the notification of the operated device which is positive iff the device's actuator is not out of order, the position of the devices whose position detector is normal, as well as the fault status of the devices whose fault detector is normal, or their negation for those whose fault detector lies.

A pure PDDL encoding of the actions can be obtained by simulating GPT's ramification rules via an extra action treating fed devices and affected breakers, which can be easily constrained to interleave with the other actions. Another option is to add a parameter `?c` to the derived predicates, and make their value conditional upon device `?c` being closed. This is achieved by replacing `(closed ?x)` in the present definitions with `(Cclosed ?c ?x)` defined to be true when `?x` is closed or `?c = ?x`. The actions' effects are then easily described using the conditional version of `affected`.

Goal

Since the goal is to address the full scope of the benchmark for which there is no specified goal but the request to minimize the breakdown cost, the problem is formulated as a pure optimization problem. Thus, the proposition `done` is initially declared false, and an action `finish` which makes `done` true and whose cost is linear in the number of unsupplied non-faulty lines is made available; see Fig. 9.5. Note that all other actions have the default cost of 1. Setting the goal to `done` allows GPT to finish the plan at any step by paying the price corresponding to the current breakdown costs. Therefore, any optimal policy found by GPT for the goal `done` minimizes breakdown costs.

It is important to understand that minimum breakdown costs cannot be achieved by supplying GPT with a restoration goal such as "supply all lines that can be supplied". The reason is that partial observability sometimes prevents the existence of a policy satisfying such a goal, even though actions can still be taken to reduce breakdown costs.

For a small example, consider the linear network shown in Fig. 9.6 with 3 lines, one circuit-breaker at each end, and two switches. Consider initially that all devices are closed, except `SD2` which is open, there is at least one fault on the area fed by `CB1`, all fault detectors are out of order, the position detector of `CB2` is out of order, and everything else is correct. Under the observation that `CB1` has opened, then there are three equiprobable hypothesis: line `l1` is faulty, line `l2` is faulty, or lines `l1` and `l2` are faulty simultaneously. Since there is a total lack of information from the network, there is no way of testing which hypothesis is true. On the other hand, the policy that opens the switch `SD1` and closes `CB1` achieves respectively a number of 1, 0, and 0 unsupplied non-faulty lines in each of the three scenarios for a expected number $1/3$ of unsupplied non-faulty lines. Clearly, no policy achieves zero expected number, so the above policy is optimal.

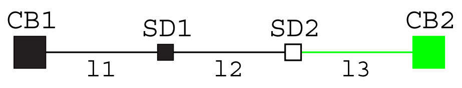


Figure 9.6: Small PSR example used to show that the question of which lines can be supplied is undecidable (see text).

Comparison with $\mathcal{AR}$ encodings

The encoding of PSR differs substantially from the $\mathcal{AR}$ encodings used by MBP [15], which are propositional in nature and are automatically generated for a given network by a custom procedure. The procedure computes all minimal acyclic paths in the network and uses those to determine all the conditions on device positions and line modes under which a given device is affected by an action. Unfortunately, the number of these conditions and consequently the action description can grow exponentially large in the number of devices in the network. This together with the propositional character of $\mathcal{AR}$ leads to huge descriptions (over 8 MB for the rural network in Fig. 9.3), which is to be contrasted with the above concise network-independent encoding. On the other hand, the number of propositions in the $\mathcal{AR}$ encoding is smaller than that induced by the present one, and this positively impacts the efficiency of BDD-based planners such as MBP. Indeed, it was found that a propositional expansion of the present encoding into $\mathcal{AR}$ caused computational difficulties for MBP and that vice versa the $\mathcal{AR}$ encoding generated by the procedure was unmanageable for GPT.

The goal constitutes another difference between the two encodings. Since MBP does not reason about plan cost, [15] treats PSR as the problem of finding a contingent plan achieving the goal of supplying all suppliable lines, under various additional assumptions (D1–D3) to be discussed below. As deciding which lines are suppliable in a given state is a non-trivial problem, the $\mathcal{AR}$ encoding identifies suppliable with a stricter condition of existence of a "safe path" between a breaker and the line, that is a path consisting entirely of non-faulty lines and maneuverable devices (i.e. with normal actuators). As it was argued above, GPT is able to act optimally even when no such plan exists. It seems appropriate to emphasize that the encoding given here is targeted at solving a typically much harder problem, i.e. one requiring optimization.

9.4.3 Experimental Results

The experiments involve several variants of the full PSR given above, the comparison of several heuristics and algorithms for computing them, and two optimization criteria.

Domains

The following modifications to the full PSR domain, explained in Figures 9.4 and 9.5, are considered.

D1. To prevent closing actions of power loops or the creation of areas fed by multiple breakers, the precondition `(:not (bad ?x))` is added to the `(close ?x)` action where `bad` is true for switches fed on two sides, and for breakers connected to a loop in the network.

D2. To prevent opening devices that are currently fed, the precondition `(:not (fed ?x))` is added to the `(open ?x)` action. Also, close actions are preferred over open actions when breaking ties.

D3. A variation where the goal is to supply all lines that can be supplied (like in MDP), where the 'suppliable' lines are computed with a notion of existence of safe paths as explained above.

D4. A variation where a fault detector of a unfed device returns no information rather than the same information it was returning when last fed.

Modifications D1–D3 lead GPT to address essentially the same domain as MBP, under the same assumptions as the experiments reported in [15]. This enables a somewhat fairer comparison of the strength and weaknesses of both planners, although GPT still attempts to minimize action costs while MBP does not. D1 is useful on its own,

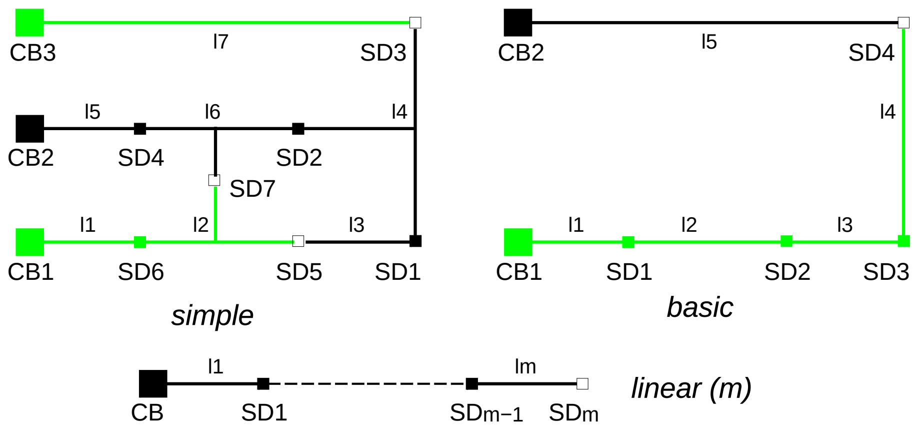


Figure 9.7: Small Test Networks

as no loops and no double feeding is a standard assumption in supply restoration. D4 enables us to address larger problem instances than would be normally possible: it obviates the need to remember fault status and make them state variables, which significantly reduces the number of states.

The experiments were performed for the following domain variations, standard (`std`): the full PSR domain with modification D1, `mbp`: the full domain with modifications D1–D3, as well as `std*` and `mbp*`, which result from applying modification D4 to `std` and `mbp`, respectively.

Algorithms

Three algorithmic settings were tried over each domain instance, `org`: the standard Labeled RTDP algorithm with a precompilation step used to compute the state space and the h_{QMDP} heuristic, `incr`: LRTDP with incremental state space generation and dynamic computation of the heuristic, and `h=0`: LRTDP with incremental state space generation seeded with $h(b) = 0$ for all belief states b.

The dynamic computation of the heuristic is performed with a IDA* search using transposition tables; see [35] for the details.

Optimization Criteria

Optimal policies with minimal expected cost (`exp`) or worst-case cost (`wst`) were produced. For the `mbp` and `mbp*` domains, where the supply restoration goal is given, the cost only includes (unit) action costs, not breakdown costs. For the `exp` criteria, all the initial states in each problem instance are taken as equiprobable, as this avoids making up fault probabilities. In principle, however, GPT can handle any distribution. In particular, it would be straightforward to specify independent fault and mode probabilities and consider the corresponding induced distribution.

Small Networks

GPT is first tested on the small networks `basic`, `simple` and `linear` in Fig. 9.7. For `basic`, 5 problem instances `b1`–`b5` are considered, where bn has at most n faulty lines at unknown locations. For `simple`, 7 instances `s1`–`s7` are considered, where sn has exactly n faulty lines at unknown locations (e.g. the PDDL encoding of `s4` is shown in Figures 9.4 and 9.5). Similarly, lm_en is an instance of the linear network with m lines and exactly n faulty lines – 12 such instances are considered. All experiments with small networks were run on a standalone Ultra-10 with 300MB of memory and a clock speed of 440MHz.

Tables 9.4 and 9.5 show the results obtained for the `linear` networks for the `mbp` and `std` variation respectively. Similarly, Tables 9.6 and 9.7 show the results for the `basic` and `simple` networks. In all cases, the results are for GPT with incremental state space generation and the dynamic version of h_{QMDP} (`incr` algorithm). The left-hand sides of the tables refer to the non-starred versions of the domains, and the right-hand side to the starred versions explained above. The tables show the run time in seconds, the number of states generated, and the optimal policy cost. Note that the cost of `mbp*` and `std*` policies are incomparable.

The original version of GPT (`org` algorithm) ran out of memory during the precompilation step for basically all but the tiniest instances. As shown in the figures, the `incr` algorithm is at least able to cope with all instances of the * domain versions which lack the additional exponential growth of the number of reachable states. The price to pay for the time and memory gain provided by the * versions is a reduction in policy quality. For instance, even in a very simple instance such as `l4_e2`, the cost of the optimal policy for `mbp/exp` (resp. `std/exp`) is 2.50 (resp. 9.16), and that for `mbp*/exp` (resp. `std*/exp`) is 3.00 (resp. 9.50).

Another observation is that the run times for `simple` and `linear`, for which the *exact* number of faults is known, exhibit an easy-hard-easy pattern. The same phenomenon was observed in [15], and was attributed to the ability of MBP's symbolic algorithm to exploit problem structure. Since GPT's algorithm enumerates states and is unable to exploit such structure, it appears that the real cause for the pattern is that the number of states is dictated by the number of ways of choosing n faulty lines among m, which peaks at $n = m/2$. Also note that critically constrained `std` worst-case instances are much easier to solve optimally than their expected-case counterpart. This is because when there are at least as many faults as breakers, in the worst case nothing is resuppliable and so the optimal policy is to do nothing, while in the expected case the policy must still prescribe what to do for other situations besides the worst.

Sticking with incremental state generation and * versions, Table 9.8 evaluates the benefits of the heuristic (`incr` algorithm) in comparison to $h = 0$. The run time improvement is dramatic, up to the point where even some `std*/exp` instances are not solvable with the zero heuristic. Similar results are obtained with the other domains and criteria.

Larger Networks

After testing GPT on small instances, the challenging `rural` network solved (non-optimally) by MBP was considered. In the original instance, see [15], everything about the rural network in Fig. 9.3 is known to be correct, except the mode of lines `l3` and `l15`, the mode of the fault detectors of `SD1`, `SD2`, `SD3`, `S26`, as well as the mode of the position detector and actuator of `SD26` which are unknown. This leads to 1944 initial states and to a myriad of reachable states ($\simeq 10^{20}$ for `std`).

Unfortunately, even equipped with the heuristic, GPT is unable to solve this instance. To identify the largest scaled down version of this problem that GPT could solve, two variations were considered. `simplified-rural` is like `rural` except that all intermediate switches on areas other than that fed by `CB1` are removed. On those areas, only the breaker and open switches are kept, which leaves 7 breakers, 11 lines, and 11 switches. `small-rural` is further reduced by removing all breakers except `CB1`, `CB5` and `CB6` as well as the area attached to them, so are left with 3 breakers, 7 lines, and 6 switches. The exact layouts of the two networks are shown in Fig. 9.8.

For both networks, 8 problem instances were considered with an increasing degree of uncertainty peaking at that of the original. These instances comprise 2, 4, 8, 24, 72, 216, 648, and 1944 initial states, respectively. Fig. 9.3 shows the run time in seconds, the optimal policy cost, and number of states generated by the `incr` algorithm for `mbp*` and `std*` under the two optimization criteria. The experiments were run in a time-shared server with a similar processor as before but with 4GB of memory, only half of which GPT was allowed to use.

The incremental state generation and dynamic computation of h_{QMDP} pays off again: the `org` and `h=0` algorithms were unable to solve any of those instances within the allocated memory. Although GPT is still not able to solve the larger instances, it is worth noting that some of those it can solve are *big*. Even the subset of states generated with h_{QMDP} grows near to a million, which is very significant for optimal planning in a partially observable domain. It is clear that we are reaching the limits of what can be achieved with algorithms based on explicit state representations, and that the use of compact representations is the key to further improvement of these results. Symbolic representations are indeed one of the reasons behind the success of the MBP planner: it solved the original rural network instance in a few seconds following a 30min compilation of the network description into efficient data structures. Another obvious reason behind the discrepancy between the run-times of MBP and GPT on similar problems is that GPT always optimizes cost. E.g., on small networks, this causes the `mbp*` domain to be only somewhat easier for

	regular versions			* versions		
prob.	time	cost	states	time	cost	states
	`mbp/wst/incr`			`mbp*/wst/incr`		
`l4_e0`	0.14	0.00	2	0.15	0.00	2
`l4_e2`	0.34	4.00	507	0.30	4.00	310
`l4_e4`	0.14	0.00	7	0.15	0.00	7
`l6_e0`	0.14	0.00	2	0.15	0.00	2
`l6_e2`	4.15	5.00	5,838	2.74	5.00	3,152
`l6_e4`	3.20	4.00	6,073	1.60	4.00	2,418
`l6_e6`	0.14	0.00	9	0.17	0.00	9
`l8_e0`	0.14	0.00	2	0.19	0.00	2
`l8_e2`	57.40	5.00	42,167	30.43	5.00	20,880
`l8_e4`	—	—	—	139.04	6.00	55,383
`l8_e6`	26.54	4.00	30,020	12.47	4.00	12,255
`l8_e8`	0.19	0.00	11	0.19	0.00	11
	`mbp/exp/incr`			`mbp*/exp/incr`		
`l4_e0`	0.26	0.00	2	0.32	0.00	2
`l4_e2`	0.55	2.50	525	0.52	3.00	316
`l4_e4`	0.27	0.00	7	0.28	0.00	7
`l6_e0`	0.27	0.00	2	0.29	0.00	2
`l6_e2`	6.71	3.20	6,950	4.31	3.66	3,316
`l6_e4`	7.67	2.86	10,716	4.69	3.20	3,056
`l6_e6`	0.27	0.00	9	0.31	0.00	9
`l8_e0`	0.28	0.00	2	0.32	0.00	2
`l8_e2`	100.89	3.57	56,126	52.59	4.00	24,902
`l8_e4`	—	—	—	236.28	3.97	59,376
`l8_e6`	278.88	2.75	136,024	47.79	2.92	22,334
`l8_e8`	0.34	0.00	11	0.32	0.00	11

Table 9.4: Results for the `linear` network under the `mbp` encodings with different cost optimization criteria. The problem is solved using the `incr` version of the algorithm. A dash (—) means that GPT ran out of memory. All times are in seconds.

	regular versions			* versions		
prob.	time	cost	states	time	cost	states
	`std/wst/incr`			`std*/wst/incr`		
`l4_e0`	0.15	0.00	53	0.17	0.00	53
`l4_e2`	0.51	10.00	1,399	0.30	10.00	549
`l4_e4`	0.14	0.00	42	0.18	0.00	42
`l6_e0`	0.17	0.00	86	0.16	0.00	86
`l6_e2`	18.22	20.00	25,360	3.35	20.00	5,110
`l6_e4`	24.26	10.00	46,695	2.15	10.00	4,526
`l6_e6`	0.16	0.00	72	0.16	0.00	72
`l8_e0`	0.23	0.00	123	0.22	0.00	123
`l8_e2`	—	—	—	60.11	30.00	37,718
`l8_e4`	—	—	—	109.97	20.00	85,452
`l8_e6`	—	—	—	22.41	10.00	31,157
`l8_e8`	0.23	0.00	110	0.19	0.00	110
	`std/exp/incr`			`std*/exp/incr`		
`l4_e0`	0.28	0.00	53	0.30	0.00	53
`l4_e2`	0.97	9.16	1,726	0.64	9.50	626
`l4_e4`	0.28	0.00	42	0.30	0.00	42
`l6_e0`	0.29	0.00	86	0.29	0.00	86
`l6_e2`	30.37	16.53	30,809	8.37	17.00	6,627
`l6_e4`	32.13	10.00	51,569	3.34	10.00	5,385
`l6_e6`	0.29	0.00	72	0.29	0.00	72
`l8_e0`	0.37	0.00	123	0.35	0.00	123
`l8_e2`	—	—	—	151.65	24.00	50,538
`l8_e4`	—	—	—	521.30	19.42	118,752
`l8_e6`	—	—	—	24.11	10.00	31,907
`l8_e8`	0.37	0.00	110	0.33	0.00	110

Table 9.5: Results for the `linear` network under the `std` encodings with different cost optimization criteria. The problem is solved using the `incr` version of the algorithm. A dash (—) means that GPT ran out of memory. All times are in seconds.

prob.	regular versions time	cost	states	* versions time	cost	states
	mbp/wst/incr			mbp*/wst/incr		
b1	0.58	5.00	1,065	0.34	5.00	445
b2	2.16	5.00	3,992	1.00	5.00	1,341
b3	4.31	6.00	7,091	1.85	6.00	2,115
b4	4.77	6.00	7,952	2.07	6.00	2,452
b5	4.40	6.00	7,862	2.05	6.00	2,510
s1	46.86	5.00	28,987	9.28	5.00	5,552
s2	—	—	—	42.72	8.00	22,720
s3	—	—	—	105.72	9.00	36,689
s4	—	—	—	107.89	8.00	31,489
s5	354.60	6.00	198,621	26.30	6.00	14,909
s6	7.51	4.00	10,935	2.35	4.00	2,862
s7	0.17	0.00	13	0.18	0.00	13
	mbp/exp/incr			mbp*/exp/incr		
b1	0.80	3.00	1,065	0.52	3.16	466
b2	2.39	3.31	3,689	2.22	3.87	1,377
b3	6.15	3.38	6,905	4.23	3.84	2,178
b4	7.85	3.32	7,825	4.16	3.74	2,532
b5	8.67	3.31	8,469	4.68	3.71	2,599
s1	47.53	3.28	28,987	8.70	3.42	5,155
s2	—	—	—	42.25	5.04	22,091
s3	—	—	—	103.28	5.42	36,312
s4	—	—	—	247.63	5.31	32,190
s5	—	—	—	115.31	4.42	16,775
s6	30.14	2.42	31,614	6.45	3.00	3,967
s7	0.30	0.00	13	0.33	0.00	13

Table 9.6: Results for the `basic` and `simple` networks under the `mbp` encodings with different cost optimization criteria. The problem is solved using the `incr` version of the algorithm. A dash (—) means that GPT ran out of memory. All times are in seconds.

	regular versions			* versions		
prob.	time	cost	states	time	cost	states
	`std/wst/incr`			`std*/wst/incr`		
`b1`	1.13	5.00	2,572	0.60	5.00	1,096
`b2`	5.41	15.00	12,200	0.89	15.00	2,192
`b3`	10.21	15.00	24,005	1.30	15.00	3,378
`b4`	10.74	15.00	26,011	1.45	15.00	3,862
`b5`	10.96	15.00	26,022	1.48	15.00	3,916
`s1`	61.58	4.00	44,412	16.68	5.00	11,174
`s2`	—	—	—	75.74	9.00	37,523
`s3`	—	—	—	58.48	20.00	44,735
`s4`	—	—	—	45.87	15.00	39,069
`s5`	—	—	—	17.79	10.00	20,137
`s6`	29.69	5.00	48,444	2.45	5.00	4,736
`s7`	0.22	0.00	123	0.18	0.00	123
	`std/exp/incr`			`std*/exp/incr`		
`b1`	1.12	3.00	2,415	0.72	3.16	1,039
`b2`	9.31	6.43	15,894	3.17	7.00	2,891
`b3`	26.87	7.23	33,131	9.90	7.69	4,522
`b4`	30.53	7.03	37,691	7.97	7.32	5,219
`b5`	33.48	6.90	39,373	8.50	7.12	5,252
`s1`	70.48	3.28	50,222	13.14	3.42	9,376
`s2`	—	—	—	62.86	5.28	36,484
`s3`	—	—	—	175.24	8.85	61,411
`s4`	—	—	—	379.12	10.68	60,145
`s5`	—	—	—	40.96	8.09	28,388
`s6`	29.95	4.28	48,444	2.99	4.28	5,288
`s7`	0.33	0.00	123	0.31	0.00	123

Table 9.7: Results for the `basic` and `simple` networks under the `std` encodings with different cost optimization criteria. The problem is solved using the `incr` version of the algorithm. A dash (—) means that GPT ran out of memory. All times are in seconds.

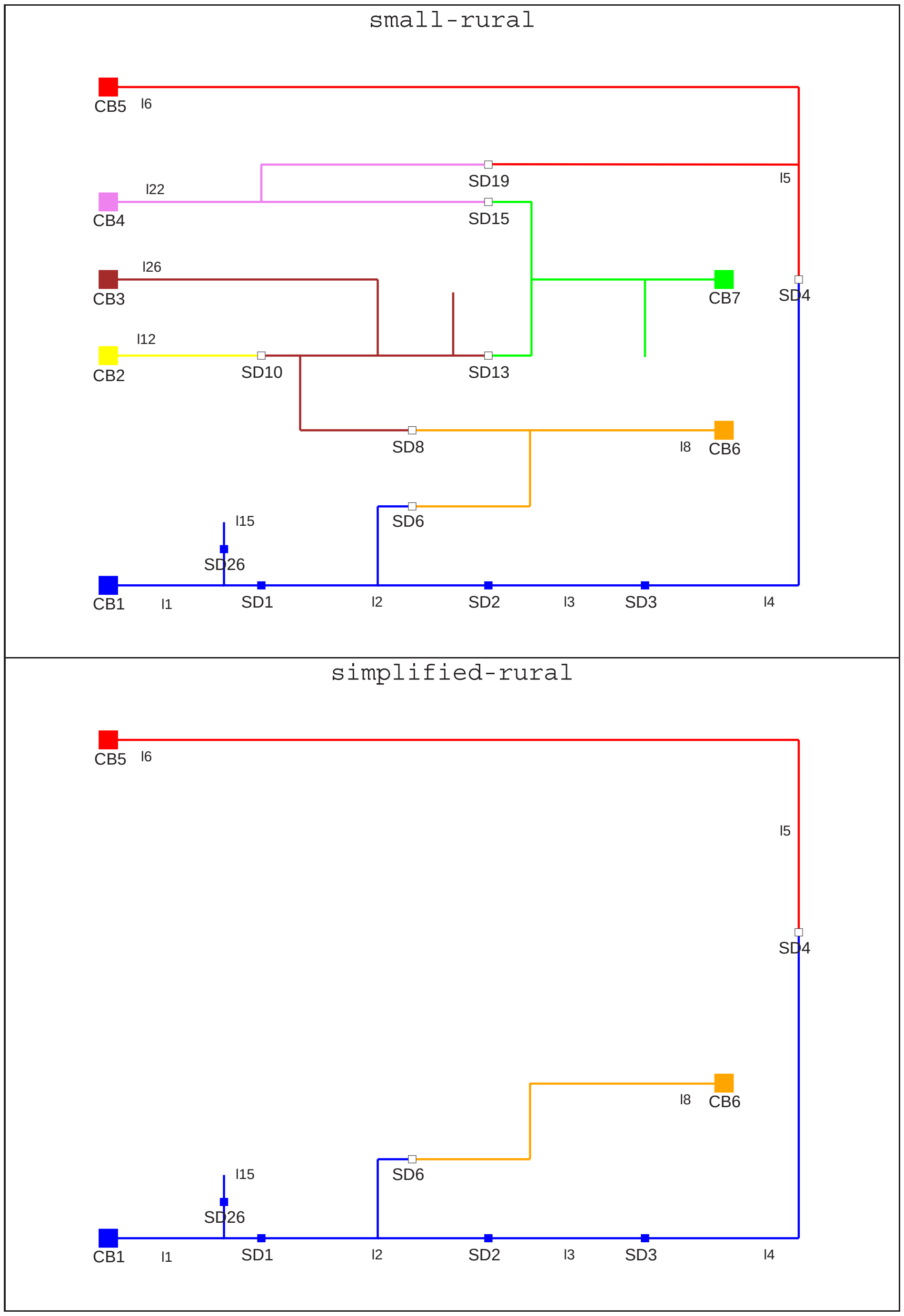


Figure 9.8: Networks layout for the `small-rural` and `simplified-rural` networks.

	* version	
problem	`incr`	`h=0`
	`std*/exp`	
`l4_e0`	0.30	0.29
`l4_e2`	0.64	3.07
`l4_e4`	0.30	0.30
`l6_e0`	0.29	0.27
`l6_e2`	8.37	139.80
`l6_e4`	3.34	177.88
`l6_e6`	0.29	0.28
`l8_e0`	0.35	0.28
`l8_e2`	151.65	3,683.23
`l8_e4`	521.30	—
`l8_e6`	24.11	—
`l8_e8`	0.33	0.31

	* version	
problem	`incr`	`h=0`
	`std*/exp`	
`s1`	13.14	58.34
`s2`	62.86	1,473.63
`s3`	175.24	12,054.37
`s4`	379.12	14,785.79
`s5`	40.96	3,322.15
`s6`	2.99	194.88
`s7`	0.31	0.29

Table 9.8: Run Times for incremental version and $h = 0$ on small instances. A dash (—) means that GPT ran out of memory. All times are in seconds.

GPT than its `std*` counterpart.

9.5 Summary and Notes

Most of the ideas presented so far in this dissertation are implemented in the GPT system. This chapter presents a study of the benefits and limitation of GPT on different domains. The domains were chosen to confront GPT with three important difficulties: domains with deep optimal solutions, domains with shallow solutions but large branching factor and non-informative heuristics, and domains with shallow solutions and informative heuristics but huge numbers of states.

As it was shown, GPT is able in all cases to solve a number of instances but its scalability is limited, yet the author is not aware of any other better method to optimally solve these problems; e.g. better (domain-dependent) admissible heuristics. Although the first two domain are not of much practical importance, the PSR domain is very close to a real problem of practical importance. The PSR domain seems to be very interesting as a comprehensive benchmark since it allows for multiple instances with different degrees of complexity, non-obvious and interesting solutions, and a lot of meaningful and interesting variations.

PSR has been addressed by others in [15], yet they were not interested in optimality (not even in reducing break-down cost), so the solution times and plan quality are not comparable. However, they were able to cope with larger problem instances very fast by pre-compiling the theory into propositional form and then using a model checker.

GPT has been confronted with other problems in the past. For example, with optimal synthesis of sorting algorithms and decision trees in [24], robot navigation, the omelette problem and an assembly line problem in [91, 29], medical diagnosis in [26], etc.

Among these, perhaps the most interesting are the synthesis of optimal sorting algorithms and decision trees.

In the sorting domain, the goal is to perform comparisons and swaps between elements of an array to achieve a configuration where all elements are sorted in ascending order from an initial configuration where no information about the order of the elements is known. Thus, the optimal policy for this problem is an optimal sorting algorithm tailored for the specific number of elements. [24] presents optimal solutions for $n = 5$, and suboptimal ones, using a non-admissible heuristic, for $n = 10$.

In the decision tree domain, the goal is to compute a decision tree that minimizes the classification error over a training set. The available actions to the agent are either to test (observe) the value of a discrete attribute (data field), or to assign a class label. Since each action has an associated cost, the algorithm tradeoffs training error against

		simplified_rural				
		rsp1	rsp2	rsp3	rsp4	rsp5
mbp*/wst/incr	time	8.56	240.88	481.05	9,339.97	—
	cost	3.00	6.00	6.00	8.00	—
	states	729	20,673	38,646	639,947	—
mbp*/exp/incr	time	8.53	231.79	460.43	1,922.22	15,384.11
	cost	2.00	3.75	3.75	4.17	4.17
	states	729	20,103	37,429	148,129	890,097
std*/wst/incr	time	80.86	2,039.79	3,794.65	—	—
	cost	3.00	6.00	6.00	—	—
	states	6929	148,631	279,134	—	—
std*/exp/incr	time	82.20	1,836.27	3,352.00	—	—
	cost	2.00	3.75	3.75	—	—
	states	6929	131,668	243,421	—	—

Table 9.9: Results for the simplified_rural network and the incremental version. A dash (—) means that GPT ran out of memory. All times are in seconds.

complexity of the decision tree. Since complex decision trees tend to overfit the data, the model is able to tradeoff training error against overfit. The results in [24] show that the resulting decision trees achieve better generalization error than standard algorithms like C4.5 [165]. [11] also address the task of inferring decision trees using MDP and POMDP models.

		small_rural							
		rsm1	rsm2	rsm3	rsm4	rsm5	rsm6	rsm7	rsm8
mbp*/wst/incr	time	0.59	2.74	5.42	24.57	108.54	583.38	3,950.55	36,196.09
	cost	3.00	6.00	6.00	8.00	8.00	8.00	9.00	9.00
	states	457	2,096	4,099	10,082	30,322	91,015	273,386	822,475
mbp*/exp/incr	time	0.74	2.47	4.82	19.69	73.73	388.63	3,153.21	81,541.57
	cost	2.00	3.75	3.75	4.17	4.17	4.17	4.40	5.19
	states	457	1,843	3,679	9,491	28,666	84,694	266,034	822,026
std*/wst/incr	time	1.26	4.97	9.04	28.81	114.49	574.59	7,076.00	—
	cost	3.00	6.00	6.00	11.00	11.00	11.00	12.00	—
	states	1138	4,519	8,225	19,880	61,427	183,577	615,771	—
std*/exp/incr	time	1.45	3.86	7.23	33.53	132.28	647.39	6,602.78	—
	cost	2.00	3.75	3.75	5.83	5.83	5.83	6.06	—
	states	1138	3,414	6,473	19,587	56,668	168,541	528,050	—

Table 9.10: Results for the small_rural network and the incremental version. A dash (—) means that GPT ran out of memory. All times are in seconds.

Chapter 10

Order-of-Magnitude Approximations

The mathematicals models used to formalize sequential decision making under uncertainty and partial information combine different ingredients that can be naturally divided in two classes:

(a) qualitative ingredients that define the *structure* of the problem, e.g. the set of world configurations (state space), the set of actions, the feedback that can be received, etc; and

(b) quantitative ingredients (also known as parametrizations of the structure) that, together with the qualitative information, defines the model. Examples of the quantitative information are the transition probabilities of going from one state to another after the application of actions, the action costs, etc.

In general, the optimal solutions to the induced formal models depend on both types of information. However, when trying to model real-world problems it is often the case that only the qualitative ingredients are known while only rough estimates of the quantitative information can be obtained. In such cases, the standard algorithms cannot be applied unless the missing information is 'completed' – a process that is often arbitrary and, more important, unnecessary for obtaining reasonable solutions.

This chapter defines a well-founded mathematical framework for modeling and solving such partially specified problems without committing to specific parametrizations.[1] Thus, the main goal is to develop a theory that parallels the standard theory of MDPs and POMDPs when only precise knowledge of the qualitative information is available. The resulting mathematical models, which are qualitative versions of decision processes, are called Qualitative MDPs (QMDPs) and Qualitative POMDPs (QPOMDPs).

As it will be seen, the resulting models can be thought as a generalization of the standard theories in the sense that as the quantitative information becomes more precise, the qualitative processes become 'closer' to the standard processes as well as their solutions.

More precisely, a theory using order-of-magnitude approximations based on *kappa rankings* that preserves the benefits from the standard theory of Markov Decision Processes is pursued. Kappa rankings constitute a widely accepted formalism for representing and reasoning about qualitative information [189, 159, 97], and has close ties with other approaches as for example Possibilistic Theory [70], ϵ-semantics for Default Reasoning [157, 89], and Fuzzy Logic [202].

Pearl [158], and later Wilson [201] showed that reasoning with kappa rankings amounts to reasoning with probabilities specified up to an order-of-magnitude of an unspecified quantity ε, which in turn translates into reasoning with *polynomials* over the dummy variable ε.

Therefore, the present approach is directed towards making the right definitions and characterizations using a qualitative version of the Bellman's equations, and showing that the value iteration algorithm can be used to solve such problems.

[1]Other approaches for solving unspecified problems are those based on reinforcement learning; they can be understood as methods that estimate the missing quantitative information from experimentation [190].

This chapter is a revised version of [34] and is organized as follows. In the next section, the formal mathematical objects upon which the theory is built are presented. Sections 10.2 and 10.3 develop the qualitative version of the decision processes while Section 10.4 gives examples of such processes. The chapter ends with a brief summary and discussion.

10.1 Mathematical Foundations

The approach to a qualitative theory for MDPs and POMDPs is based on the qualitative decision theory proposed by Wilson few years ago [201]. Wilson's theory, which is built upon the ideas of Pearl and Goldszmidt [159, 158, 96], defines a set of abstract quantities called *extended reals*, denoted by $\mathcal{Q}$, that are used to represent qualitative probabilities and utilities.

Each extended real is a *rational function* p/q where p and q are polynomials in ε with rational coefficients. Plainly, ε is thought as a very small but unknown quantity and the extended reals ought to be interpreted as 'information up to ε precision.' For example, quantities like $1-\varepsilon$ and ε might be used for qualitative probabilities as 'likely' and 'unlikely', and ε^{-1} for a high utility. These quantities are then combined using standard arithmetic operations between polynomials to compute expected qualitative utilities. The resulting quantities rank the different possible scenarios using a linear order $\succeq$ on $\mathcal{Q}$ [201].[2]

In order to give a qualitative version of MDPs on top of Wilson's extended reals, it is required that equations like Bellman's equations be defined and solvable. In particular, a notion of 'convergence' is needed. It is not hard to realize, that not only polynomials but infinite series on ε are needed in order to guarantee the solvability of such equations.

10.1.1 A Complete Extension of $\mathcal{Q}$

The first goal is to define a proper notion of convergence and close the set of rational functions $\mathcal{Q}$ with respect to convergence. The following development is a somewhat standard process in analysis.

Let $\mathcal{S}_\rho$ be the set of *two-sided infinite formal series* in ε with real coefficients $s = \sum_k a_k \varepsilon^k$ such that

$$\|s\|_\rho \overset{\text{def}}{=} \sum_k |a_k|\,\rho^{-k} \;<\; \infty$$

where ρ is a positive real and the summation is over all integers. A sequence $\{s_n\}_n$ is said to *converge* to $s \in \mathcal{S}_\rho$ iff $\|s - s_n\|_\rho \to 0$ as $n \to \infty$, and that the sequence is *fundamental* iff $\|s_n - s_m\|_\rho \to 0$ as $n, m \to \infty$. Note that there is no a priori reason for a fundamental sequence to converge to something in $\mathcal{S}_\rho$. Fortunately, this is not the case and it can be shown that $(\mathcal{S}_\rho, \|\cdot\|_\rho)$ is a *complete* normed space; i.e. a Banach space [A12].[3] Completeness means that every fundamental sequence converges to a series in $\mathcal{S}_\rho$.

Lemma 52 *For any series s, and sequence $\{s_n\}_n$ of series, $s_n \to s$ if and only if $s_n(k) \to s(k)$ for all integers k.*

Proof: Immediate since the topology induced by the norm $\|\cdot\|_\rho$ is the topology of pointwise convergence [A8]; see [174]. □

The following arithmetic operations over series are defined in the standard way:

$$\begin{aligned}
(s+t)(k) &\overset{\text{def}}{=} s(k) + t(k), \\
(\alpha s)(k) &\overset{\text{def}}{=} \alpha s(k), \\
(s \cdot t)(k) &\overset{\text{def}}{=} \sum_{i,j} [\![i + j = k]\!] s(i) t(j) \;=\; \sum_i s(i)\, t(k-i)
\end{aligned}$$

[2]Briefly, Wilson defines $f \succeq g$ iff there exists $\xi > 0$ such that $f(\varepsilon) - g(\varepsilon) \geq 0$ for all $\varepsilon \in (0, \xi)$.

[3]Indeed, $(\mathcal{S}_\rho, \|\cdot\|_\rho)$ is the $L_1(\mu_\rho)$ space with respect to the measure μ_ρ over the integers defined by $\mu_\rho\{k\} = \rho^{-k}$. The Riesz-Fischer Theorem asserts that this space is complete; see [174].

where $s(k)$ is the functional notation for the kth term of s, and $[\![\varphi]\!]$ is 1 (resp. 0) if φ is true (resp. false).

Two important series are **0** and **1** that are defined as $\mathbf{0}(k) \overset{\text{def}}{=} 0$, $\mathbf{1}(0) \overset{\text{def}}{=} 1$ and $\mathbf{1}(0) \overset{\text{def}}{=} 0$ for all integer $k \neq 0$. They are the identity element for the sum and product of series respectively. The *order of magnitude* of $s \in \mathcal{S}_\rho$ is defined as $s^\circ \overset{\text{def}}{=} \inf\{k \in \mathbb{Z} : s(k) \neq 0\}$ while the order of magnitude of the **0** series is defined as ∞. If $\underline{\mathcal{S}}_\rho$ is defined as the collection of series from $\mathcal{S}_\rho$ with order of magnitude $> -\infty$, then

Theorem 53 *$\underline{\mathcal{S}}_\rho$ is a field; i.e. $(\underline{\mathcal{S}}_\rho, \mathbf{0}, +)$ is a commutative group, $(\underline{\mathcal{S}}_\rho \setminus \{\mathbf{0}\}, \mathbf{1}, \cdot)$ is a commutative group, and the product is distributive with respect to the sum.*

Proof: It is easy to prove that the sum and product of series are associative and commutative operations, and that the product is distributive with respect to the sum. Also, **0** is the additive identity and $-s$, defined as $(-s)(k) = -s(k)$, is the additive inverse of s. Thus, $(\mathcal{S}_\rho, +)$ is a commutative additive group. The series **1** is the multiplicative unit. To define the multiplicative inverse s^{-1}, set

$$s^{-1}(-s^\circ + k) = \begin{cases} 0 & \text{if } k < 0, \\ \frac{1}{s(s^\circ)} & \text{if } k = 0, \\ -\frac{1}{s(s^\circ)} \sum_{j=1}^{k} s(s^\circ + j)\, s^{-1}(-s^\circ + k - j) & \text{if } k > 0. \end{cases}$$

Obviously, $\left(s^{-1}\right)^\circ = -s^\circ$, and

$$\begin{aligned} (s \cdot s^{-1})(0) &= \sum_j s(j) s^{-1}(-j) = s(s^\circ)\, s^{-1}(-s^\circ) = 1, \\ (s \cdot s^{-1})(k) &= \sum_j s(j)\, s^{-1}(-j + k) = s(s^\circ)\, s^{-1}(-s^\circ + k) + \sum_{j > s^\circ} s(j)\, s^{-1}(-j + k) \\ &= -\sum_{j=1}^{k} s(s^\circ + j)\, s^{-1}(-s^\circ + k - j) + \sum_{j > s^\circ} s(j)\, s^{-1}(-j + k) \;=\; 0. \end{aligned}$$

Therefore, $(\mathcal{S}_\rho \setminus \{\mathbf{0}\}, \cdot)$ is a multiplicative group. □

Example. Let $s = \sum_{k \geq 0} 2^{-k} \varepsilon^k$. Then, its multiplicative inverse is $s^{-1} = 1 - \frac{1}{2}\varepsilon$. To check this, for $k > 0$,

$$\begin{aligned} (s \cdot s^{-1})(-k) &= 0, \\ (s \cdot s^{-1})(0) &= s(0) s^{-1}(0) \;=\; 1, \\ (s \cdot s^{-1})(k) &= \sum_i s(i) s^{-1}(k - i) \\ &= s(k-1) s^{-1}(1) + s(k) s^{-1}(0) \\ &= -2^{-k+1} 2^{-1} + 2^{-k} \;=\; 0. \end{aligned}$$

Hence $s \cdot s^{-1} = \mathbf{1}$. □

It is not hard to show that the set $\mathcal{Q}$ of extended reals is a dense set in $\mathcal{S}_\rho$ [A6], i.e. that for any $s \in \mathcal{S}_\rho$ there exists a sequence $\{s_n\}_n$ from $\mathcal{Q}$ that converges to s. This fact is used to induce a linear order in $\underline{\mathcal{S}}_\rho$ from the linear order in $\mathcal{Q}$.

10.1.2 Linear Order

The construction is done in the standard way by defining the set $\mathcal{P}$ of positive elements in $\underline{\mathcal{S}}_\rho$. Denote with $\succeq$ the order in $\mathcal{Q}$ and let $s \in \underline{\mathcal{S}}_\rho$ be different from **0**. Since $\mathcal{Q}$ is dense, there exists a sequence $\{s_n\}_n$ from $\mathcal{Q}$ that converges to s. Moreover, the sequence can be chosen so that $s_n^\circ \leq s^\circ$ for all n. Then, the series is by definition in $\mathcal{P}$ if and only if there exists a sequence $\{s_n\}_n$ of elements in $\mathcal{Q}$ with $s_n^\circ \leq s^\circ$ and an integer N such that $s_n \succ \mathbf{0}$ for all $n > N$. The following theorem establishes that $\mathcal{P}$ is well defined and that it satisfies the desired properties for a linear order.

Theorem 54 *Let $s \in \underline{\mathcal{S}}_\rho$ be different from $\mathbf{0}$. Then,*

(i) s is in (or not in) $\mathcal{P}$ independently of the chosen series $\{s_n\}_n$, i.e. from any two such series $\{s_n\}_n$ and $\{s'_n\}_n$ the conclusion whether s belongs to $\mathcal{P}$ is the same,

(ii) either $s \in \mathcal{P}$ or $-s \in \mathcal{P}$,

(iii) $s \in \mathcal{P}$ if and only if $s(s^\circ) > 0$.

Proof: By Lemma 52, $s_n(k) \to s(k)$ for all integer k if s_n converges to s. Then, the condition $s_n^\circ \leq s^\circ$ implies that equality actually holds for large enough n. All claims follows by observing that $s_n \succ \mathbf{0}$ if and only if $s_n(s_n^\circ) > 0$. □

Once the set of positive elements is constructed, the relational symbols are defined in the standard way; e.g. for $s, t \in \underline{\mathcal{S}}_\rho$, $s > t$ if and only if $s - t \in \mathcal{P}$.

Two important remarks are that this order cannot be consistently extended into the full set $\mathcal{S}_\rho$ and that, as the set of rational functions $\mathcal{Q}$, the field $\underline{\mathcal{S}}_\rho$ also lacks the *least upper bound* property of the reals [A2] since it contains infinitesimal elements.

The rest of this section presents versions of standard mathematical ideas involving the extended reals.

10.1.3 Normed Vector Spaces

An n-dimensional normed vector space $\mathcal{S}_\rho^n$ with elements in $\mathcal{S}_\rho$ can be given using the norm $\|J\| \overset{\text{def}}{=} \max_{i=1,\dots,n} \|J(i)\|_\rho$; see [A11]. Since $\mathcal{S}_\rho$ is complete, $\mathcal{S}_\rho^n$ is also complete; i.e. it is a Banach space [A12]. A map $T : \mathcal{S}_\rho^n \to \mathcal{S}_\rho^n$ is a contraction with coefficient $\alpha \in [0, 1)$ if $\|TJ - TJ'\| \leq \alpha\|J - J'\|$ for all $J, J' \in \mathcal{S}_\rho^n$. If so, T has a unique fixed point $J^* \in \mathcal{S}_\rho^n$ [A14].

More general vector spaces can be defined considering mappings $\mathcal{S}_\rho^X$ where X is a (finite or infinite) set. The associated norm is then $\|J\| = \sup_{x \in X} \|J(x)\|_\rho$. Thus, if $|X| = n$, $\mathcal{S}_\rho^X$ is the n-dimensional space $\mathcal{S}_\rho^n$, while if $|X| = \infty$ then $\mathcal{S}_\rho^X$ is infinite dimensional.

10.1.4 Qualitative Probabilities

An $\mathcal{S}_\rho$-probability space is a triplet $(\Omega, \mathcal{F}, P)$ where Ω is a *finite* set of outcomes, $\mathcal{F}$ is the set of all subsets of Ω, and P is a $\mathcal{S}_\rho$-valued function on $\mathcal{F}$ such that:

(i) $P(A) \geq \mathbf{0}$ for all $A \subseteq \Omega$,

(ii) $P(A \cup B) = P(A) + P(B)$ for all disjoint subsets $A, B \subseteq \Omega$, and

(iii) $P(\Omega) = \mathbf{1}$.

In this case, P is called either a $\mathcal{S}_\rho$-probability or a *qualitative probability* over Ω. A random variable X on $(\Omega, \mathcal{F}, P)$ is a mapping $\Omega \to \mathcal{S}_\rho$, and its expected value is defined as $EX \overset{\text{def}}{=} \sum_{\omega \in \Omega} X(\omega)\, P(\{\omega\})$. Compare these definition with that of standard probability theory in [A20] and [A23].

10.1.5 Kappa Rankings

A kappa ranking is a function κ that maps subsets in $\mathcal{F}$ into the non-negative integers plus ∞ such that: $\kappa(\emptyset) = \infty$, $\kappa(\Omega) = 0$ and $\kappa(A \cup B) = \min\{\kappa(A), \kappa(B)\}$ for disjoint subsets $A, B \subseteq \Omega$.

Equivalently, a kappa ranking is a function from Ω to the non-negative integers plus ∞ such that $\kappa(\omega) = 0$ for at least one $\omega \in \Omega$, and then extended over the subsets in $\mathcal{F}$ by taking minimums; i.e.

$$\kappa(A) \overset{\text{def}}{=} \min\{\kappa(\omega) : \omega \in A\}.$$

A kappa ranking is usually interpreted as a partition or ranking over worlds into degrees of *disbelief* so that values of 0,1,2, ... refer to situations deemed as ‘likely’, ‘unlikely’, ‘very unlikely’, etc. Kappa rankings have a close connection with qualitative probabilities since if P is a qualitative probability, then P° is a kappa ranking.

10.1.6 Embeddings

As it was seen, a qualitative probability P naturally defines a kappa ranking $\kappa(A) \overset{\text{def}}{=} P^\circ(A)$. The other direction is however not uniquely defined, i.e. given a ranking κ there is more than one qualitative probability P so that $P^\circ = \kappa$.

When formalizing the qualitative processes for decision making, a well-defined transformation mapping kappa rankings to qualitative probabilities, called an *embedding*, is used.

The embedding, which depends on Ω as is denoted as $\zeta : K[\Omega] \to P[\Omega]$, is the fundamental tool for translating descriptions of decision tasks in terms of kappa rankings into descriptions in the language of qualitative probabilities. Here, $K[\Omega]$ and $P[\Omega]$ stand respectively for the collections of kappa rankings and qualitative probabilities over Ω.

Given a ranking κ and a world ω, the qualitative probability assigned to ω by the embedding of κ is denoted as $\zeta_\kappa(\omega)$.

The goal is to define the embedding in such a way that equally ranked worlds receive the same mass. That is, the embedding should satisfy the property

$$\kappa(\omega) = \kappa(\omega') \implies \zeta_\kappa(\omega) = \zeta_\kappa(\omega'). \tag{10.1}$$

This equation can be understood as requiring a 'Maximum-Entropy' embedding that maps kappa rankings into consistent qualitative probabilities. The rest of this section contains a formal definition of the embedding, an example and some properties.

Let n_m be the number of worlds with rank m, i.e. $n_m \overset{\text{def}}{=} |\{\omega \in \Omega : \kappa(\omega) = m\}|$, and define $\zeta_\kappa(\omega) \overset{\text{def}}{=} \mathbf{0}$ if $\kappa(\omega) = \infty$, and

$$\zeta_\kappa(\omega) \overset{\text{def}}{=} \frac{N_{\kappa(\omega)}}{n_{\kappa(\omega)}} \left[\varepsilon^{\kappa(\omega)} - \sum_{j > \kappa(\omega)} [\![n_j \neq 0]\!] \, \varepsilon^j \right] \tag{10.2}$$

otherwise. Observe that the summation has only a finite number of non-zero terms since Ω is finite. On the other hand, the integers N_k are defined as

$$N_0 \overset{\text{def}}{=} 1, \qquad N_k \overset{\text{def}}{=} \sum_{j=0}^{k-1} [\![n_j \neq 0]\!] \, N_j.$$

Since (10.2) only depends on the rank $\kappa(\omega)$, the Maximum-Entropy property given by (10.1) is satisfied at once. The following example illustrates the embedding for a non-trivial ranking.

Example. Let $\Omega = \{a, b, c, d, e\}$ and κ be such that $\kappa(a) = \kappa(b) = 0$, $\kappa(c) = \kappa(d) = 1$ and $\kappa(e) = 5$. Then, the induced probability is given by

$$\begin{aligned}
\zeta_\kappa(a) &= 2^{-1}(1 - \varepsilon - \varepsilon^5), \\
\zeta_\kappa(b) &= 2^{-1}(1 - \varepsilon - \varepsilon^5), \\
\zeta_\kappa(c) &= 2^{-1}(\varepsilon - \varepsilon^5), \\
\zeta_\kappa(d) &= 2^{-1}(\varepsilon - \varepsilon^5), \quad \text{and} \\
\zeta_\kappa(e) &= 2\varepsilon^5.
\end{aligned}$$

Clearly,

$$\begin{aligned}
\zeta_\kappa(a) + \zeta_\kappa(b) + \zeta_\kappa(c) + \zeta_\kappa(d) + \zeta_\kappa(e) &= (1 - \varepsilon - \varepsilon^5) + (\varepsilon - \varepsilon^5) + 2\varepsilon^5 \\
&= 1 - 2\varepsilon^5 + 2\varepsilon^5 = 1.
\end{aligned}$$

Thus, ζ_κ is a qualitative probability consistent with κ. □

Similarly, it is not hard to see that if all worlds are 0-ranked by κ, then ζ_κ is just the uniform distribution over Ω. Thus, another way of understanding the above equations is that the embedding assign uniform probabilities within the different κ-strata such that the overall sum of qualitative probabilities is one.

As the reader can check, the embedding works in a bottom up fashion by defining probabilities from higher-ranked worlds to lower-ranked worlds while maintaining consistency. The following theorem shows the soundness of the embedding and gives an important property that is used later.

Theorem 55 *Let κ be a kappa ranking over Ω. Then,*

(i) ζ_κ *is a qualitative probability over* Ω*,*

(ii) $\zeta_\kappa(\omega)^\circ = \kappa(\omega)$ *for all* $\omega \in \Omega$*, and*

(iii) $\sup_\kappa \sum_{\omega\in\Omega} \|\zeta_\kappa(\omega)\|_\rho \to 1$ *as* $\rho \to \infty$*,*

where the sup is over all kappa rankings over Ω*.*

Proof: The second claim is direct from the definition. For the first, it is only necessary to show that $\sum_\omega \zeta_\kappa(\omega) = \mathbf{1}$:

$$\begin{aligned}
\sum_{\omega\in\Omega} \zeta_\kappa(\omega) &= \sum_{\omega\in\Omega} [\![\kappa(\omega) \neq \infty]\!] \frac{N_{\kappa(\omega)}}{n_{\kappa(\omega)}} \left[\varepsilon^{\kappa(\omega)} - \sum_{j>\kappa(\omega)} [\![n_j \neq 0]\!]\, \varepsilon^j\right] \\
&= \sum_{k\geq 0} [\![n_k \neq 0]\!]\, n_k \frac{N_k}{n_k} \left[\varepsilon^k - \sum_{j>k} [\![n_j \neq 0]\!]\, \varepsilon^j\right] \\
&= \sum_{k\geq 0} [\![n_k \neq 0]\!]\, N_k\, \varepsilon^k - \sum_{j=0}^{k-1} [\![n_k \neq 0, n_j \neq 0]\!]\, N_j\, \varepsilon^k \\
&= \sum_{k\geq 0} [\![n_k \neq 0]\!]\, \varepsilon^k \left[N_k - \sum_{j=0}^{k-1} [\![n_j \neq 0]\!]\, N_j\right] \\
&= [\![n_0 \neq 0]\!]\, \varepsilon^0\, N_0 \\
&= \mathbf{1}.
\end{aligned}$$

Since, being κ a genuine ranking, $n_0 > 0$. For the last claim, fix a ranking κ over Ω. Then,

$$\sum_{\omega\in\Omega} [\![\kappa(\omega) = 0]\!]\, \|\zeta_\kappa(\omega)\|_\rho \leq 1 + \sum_{j\geq 1} \rho^{-j} \leq 1 + \frac{1}{\rho-1}, \quad \text{and}$$

$$\begin{aligned}
\sum_{\omega\in\Omega} [\![\kappa(\omega) > 0]\!]\, \|\zeta_\kappa(\omega)\|_\rho &\leq \sum_{k>0} [\![n_k \neq 0]\!]\, N_k \left\|\varepsilon^k - \sum_{j>k} [\![n_j \neq 0]\!]\, \varepsilon^j\right\|_\rho \\
&\leq \sum_{k\geq 1} [\![n_k \neq 0]\!]\, N_k \sum_{j\geq k} [\![n_j \neq 0]\!]\, \rho^{-j} \\
&\leq 2^{|\Omega|} \sum_{k\geq 1} [\![n_k \neq 0]\!] \sum_{j\geq k} [\![n_j \neq 0]\!]\, \rho^{-j} \\
&\leq 2^{|\Omega|} \sum_{k\geq 1} [\![n_k \neq 0]\!] \frac{\rho^{-k}}{1-\rho^{-1}} \\
&\leq \frac{2^{|\Omega|}}{\rho-1} \sum_{k\geq 0} [\![n_k \neq 0]\!]\, \rho^{-k} \\
&\leq \frac{|\Omega|\, 2^{|\Omega|}}{\rho-1}
\end{aligned}$$

since $[\![n_k \neq 0]\!]\, N_k \leq 2^{|\Omega|}$ (left as an exercise). Therefore,

$$\sum_{\omega\in\Omega} \|\zeta_k(\omega)\|_\rho \leq 1 + \frac{1 + |\Omega|\, 2^{|\Omega|}}{\rho - 1} \to 1$$

as $\rho \to \infty$ and independently of κ. □

10.2 Qualitative MDPs

A Qualitative Markov Decision Process (QMDP) is an MDP in which the quantitative information is given by qualitative probabilities and costs. That is, a QMDP is a transition system characterized by

QM1. a finite set of states S,

QM2. a finite set of actions $A(s) \subseteq A$ for each state $s \in S$,

QM3. *qualitative* transition probabilities $p(s, a, s')$ of making a transition to $s' \in S$ after applying action $a \in A(s)$ in state $s \in S$,

QM4. a *qualitative* cost $g(s, a)$ of applying action $a \in A(s)$ in state $s \in S$,

QM5. a discount factor $\alpha \in [0, 1]$.

From the definition of the standard MDP processes in Ch. 3, it can be seen that the only change is in the transition probabilities and action costs, i.e. they differ in the nature of the quantitative information.

To define the cost associated with a policy $\pi = (\mu_1, \mu_2, \dots)$, consider an N-stage π*-trajectory* $\tau = (x_0, x_1, \dots, x_N)$ starting at state s where $x_0 = s$ and $p(x_k, \mu_k(x_k), x_{k+1}) > \mathbf{0}$. Each such trajectory τ has a qualitative probability and cost given by

$$P_\pi(\tau) \overset{\text{def}}{=} \prod_{k=0}^{N-1} p(x_k, \mu_k(x_k), x_{k+1}), \tag{10.3}$$

$$g_\pi(\tau) \overset{\text{def}}{=} \sum_{k=0}^{N-1} \alpha^k \, g(x_k, \mu_k(x_k)). \tag{10.4}$$

The *infinite horizon qualitative expected discounted cost* of applying policy π starting at state s is defined as

$$J_\pi(s) \overset{\text{def}}{=} \lim_{N\to\infty} \sum_\tau P_\pi(\tau)\, g_\pi(\tau) \overset{\text{def}}{=} \lim_{N\to\infty} E_\pi[g_\pi(\tau)] \tag{10.5}$$

where the sum is over all N-stage π-trajectories starting at i (a finite number of them) and the expectation is with respect to the qualitative probability (10.3) (compare with (3.3) for MDPs). In general, the limit (10.5) is not always well defined. However, when all costs $g(s, a)$ are in $\mathcal{Q}$ and $\alpha < 1$, then the limit exists and J_π is well defined. From now on, it will be assumed that this is the case.

The optimal cost-to-go starting from state s, denoted by $J^*(s)$, is $J^*(s) \overset{\text{def}}{=} \inf_\pi J_\pi(s)$. It will be desirable to show that J^* is well defined and that there exists a stationary policy μ^* such that $J^* = J_{\mu^*}$. Unfortunately, such a result seems very difficult since $\underline{\mathcal{S}}_\rho$ lacks the least upper bound property of the reals [A2]. Thus, the bar is lowered and only the existence of an optimal stationary policy together with an algorithm for computing it is obtained. That is, it will be shown that the partial order $\leq$ in $\underline{\mathcal{S}}_\rho^n$, defined as $J \leq J'$ if $J(s) \leq J(s)$ for all $s \in S$, has a unique minimum in the set $\{J_\mu : \mu \text{ a stationary policy}\}$.

The result will follow if the qualitative version of the Bellman's equations has a unique solution. Let T be the Bellman operator for the qualitative MDP. Then,

Theorem 56 *If $\alpha < 1$, there exists $\rho \geq 1$ such that T is a contraction.*

Proof: Choose ρ large enough so that

$$\gamma \overset{\text{def}}{=} \max_{s\in S} \max_{a\in A(s)} \alpha \sum_{s'\in S} \|p(s, a, s')\|_\rho < 1,$$

which exists since $A(s)$ is finite and $\alpha < 1$. Let $J, J' \in \mathcal{S}_\rho^n$. Then,

$$\|TJ - TJ'\| = \max_{s\in S} \|(TJ)(s) - (TJ')(s)\|_\rho$$

$$\begin{aligned}
&= \max_{s\in S}\left\| \left(\min_{a\in A(s)} g(s,a) + \alpha \sum_{s'\in S} p(s,a,s')J(s')\right) - \right. \\
&\qquad \left. \left(\min_{a\in A(s)} g(s,a) + \alpha \sum_{s'\in S} p(s,a,s')J'(s')\right) \right\|_\rho \\
&\le \max_{s\in S} \alpha \left\| \sum_{s'\in S} p(s,a^*,s')[J(s') - J'(s')] \right\|_\rho \\
&\le \max_{s\in S} \alpha \sum_{s'\in S} \|p(s,a^*,s')\|_\rho \, \|J(s') - J'(s')\|_\rho \\
&\le \|J - J'\| \max_{s\in S} \alpha \sum_{s'\in S} \|p(s,a^*,s')\|_\rho \\
&\le \gamma \|J - J'\|
\end{aligned}$$

where a^* is the action that minimizes the minimum term in the second equality, and the inequality $\|s\cdot t\|_\rho \le \|s\|_\rho\|t\|_\rho$ has been used (left as an exercise). □

The following consequence asserts that there is a best stationary policy that can be found with value iteration. Its proof is based on the fact that the Bellman operator is a contraction with similar arguments to the ones used for standard processes.

Corollary 57 *Assume $g(s,a) \in \mathcal{Q}$ for all $s \in S, a \in A(s)$ and $\alpha < 1$. Then, the qualitative version of Bellman's equations have a unique solution J^*. In addition, J^* can be found with value iteration, and the policy μ^* which is greedy with respect to J^* is the best stationary policy.*

Remarks. Note that the convergence of the value iteration algorithm is guaranteed by the existence of ρ, yet its precise value is not necessary when applying the algorithm. Also, the effective discount factor is α since γ can be made as close to α as desired by letting ρ go to ∞.

10.2.1 Order-of-Magnitude Specifications

A QMDP is defined to be an *order-of-magnitude specification* when the transition probabilities $p(s,a,s')$ are only known up to a compatible kappa ranking $\psi(s,a,s')$, i.e. $p(s,a,s')^\circ = \psi(s,a,s')$. In this case, the Bellman operator to consider is

$$(TJ)(s) \overset{\text{def}}{=} \min_{a\in A(s)} \left(g(s,u) \;+\; \alpha \sum_{s'\in S} \zeta_{\psi(s,a,\cdot)}(s')J(s') \right)$$

where $\zeta_{\psi(s,a,\cdot)}$ is the embedding of $\psi(s,a,\cdot)$. Clearly, Corollary 57 applies in this case and the induced QMDP problem can be solved with value iteration.

10.3 Qualitative POMDPs

A definition for Qualitative POMDPs (QPOMDP) can be obtained easily from the general POMDP formulation as follows:

QP1. A finite state space S,

QP2. a finite set of actions $A(s)$ for each state $s \in S$,

QP3. *qualitative* transition probabilities $p(s,a,s')$ of making a transition to state s' when action $a \in A(s)$ is applied in state s,

QP4. a finite set of observations $O(s,a) \subseteq O$ that may result after applying action $a \in A(s)$ in state s, QP5. *qualitative* observation probabilities $q(s, a, o)$ of receiving observation $o \in O(s, a)$ in state s after the application of $a \in A(s)$,

QP6. *qualitative* costs $g(s, a)$ associated with action $a \in A(a)$ and state s, and

QP7. a discount factor $\alpha \in [0, 1]$.

Similarly, a qualitative version of the belief-MDP can be obtained, yet a serious problem appears: the infiniteness of the belief space thwarts a suitable choice for ρ as in Theorem 56. On the other hand, good results for order-of-magnitude specifications of POMDPs can be derived as shown next.

10.3.1 Order-of-Magnitude Specifications

A QPOMDP is said to be an *order-of-magnitude specification* if the qualitative probabilities $p(s, a, s')$ and $q(s, a, o)$ are only known up to compatible kappa rankings $\psi(s, a, s')$ and $\vartheta(s, a, o)$ respectively; i.e.

$$\begin{aligned} p(s, a, s')^\circ &= \psi(s, a, s'), \\ q(s, a, o)^\circ &= \vartheta(s, a, o). \end{aligned}$$

In this case, a transition system over the set of *kappa belief states* will be defined and solved in an analogous way to the standard belief-MDP formulation. By a kappa belief state we mean a belief state that is only known up to a compatible kappa ranking. Thus, only kappa rankings over the states are considered as belief states, and the transition dynamics of kappa belief states is given by the order-of-magnitude versions of (3.16)–(3.18):

$$\begin{aligned} \kappa_a(s) &\overset{\text{def}}{=} \min_{s' \in S} \kappa(s') + \psi(s', a, s), \\ \kappa_a^o(s) &\overset{\text{def}}{=} \kappa_a(s) + \vartheta(s, a, o) - q(\kappa, a, o), \\ q(\kappa, a, o) &\overset{\text{def}}{=} \min_{s \in S} \kappa_a(s) + \vartheta(s, a, o), \\ O(\kappa, a) &\overset{\text{def}}{=} \{o \in O : q(\kappa, a, o) < \infty\}. \end{aligned}$$

As the reader can check, all these mappings are genuine kappa rankings over S, e.g. that there exists state s such that $\kappa_a(s) = 0$. The qualitative belief-MDP over kappa belief states is then defined as:

K1. A belief space K of kappa rankings over S,

K2. a set of actions $A(\kappa) \overset{\text{def}}{=} \{a \in A : \forall(s \in S)(\kappa(s) < \infty \Rightarrow a \in A(s))\}$ for each ranking $\kappa \in K$,

K3. a cost $g(\kappa, a) \overset{\text{def}}{=} \sum_{s \in S} g(s, a)\, \zeta_\kappa(s)$ for each $a \in A(\kappa)$ and $\kappa \in K$ where ζ_κ is the embedding of κ, and

K4. transitions $\kappa \rightsquigarrow \kappa_a^o$ with qualitative probability $\zeta_{q(\kappa,a,\cdot)}(o)$.

Therefore, the induced Bellman's equations over the belief-MDP are:

$$J(\kappa) \;=\; \min_{a \in A(\kappa)} \left(g(\kappa, a) \;+\; \alpha \sum_{o \in O(\kappa, a)} \zeta_{q(\kappa,u,\cdot)}(o)\, J(\kappa_a^o) \right) \tag{10.6}$$

with the obvious operator. The following results show that such equations have a unique fixed point solution which can be found with value iteration.

Theorem 58 *If $\alpha < 1$, there exists $\rho > 1$ such that the Bellman operator T is a contraction.*

Proof: Choose ρ large enough such that

$$\sup_{\kappa \in K} \sup_{a \in A(\kappa)} \alpha \sum_{o \in O(\kappa, a)} \|\zeta_{q(\kappa,a,\cdot)}(o)\|_\rho \;<\; 1.$$

The existence of ρ is guaranteed by $\alpha < 1$ and Theorem 55-(iii). Then, a proof similar to that of Theorem 56 gives the result. □

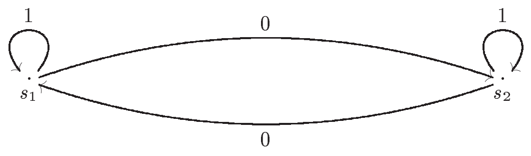


Figure 10.1: Kappa rankings for the transitions corresponding to action a_1 in the example.

Corollary 59 *Assume $g(s,a) \in \mathcal{Q}$ for all $s \in S, a \in A(s)$ and $\alpha < 1$. Then, Bellman's equations* (10.6) *have a unique solution J^*. In addition, J^* can be found with value iteration, and the policy μ^* greedy with respect to J^* is the best stationary policy.*

10.4 Examples

Consider the simple system consisting of two states $S = \{s_1, s_2\}$ and two controls a_1, a_2 such that a_1 is applicable in both states but a_2 only in s_2. The qualitative costs are $g(s_1, a_1) = 1 - \varepsilon$ and $g(s_2, a_1) = g(s_2, a_2) = \mathbf{0}$, and the kappa transition probabilities are:

$$\begin{aligned} \psi(s_1,a_1,s_1) &= 1, & \psi(s_1,a_1,s_2) &= 0, \\ \psi(s_2,a_1,s_1) &= 0, & \psi(s_2,a_1,s_2) &= 1, \\ \psi(s_2,a_2,s_1) &= \infty, \quad \text{and} & \psi(s_2,a_2,s_2) &= 0. \end{aligned}$$

Fig. 10.1 shows the kappa rankings for the transitions associated with action a_1.

10.4.1 The QMDP case

It is not difficult to check that the embeddings are:

$$\zeta_{\psi(s_1,a_1,\cdot)} = [\varepsilon, 1-\varepsilon], \quad \zeta_{\psi(s_2,a_1,\cdot)} = [1-\varepsilon, \varepsilon], \quad \text{and} \quad \zeta_{\psi(s_2,a_2,\cdot)} = [\mathbf{0}, \mathbf{1}].$$

Thus, the Bellman equations for the QMDP are:

$$\begin{aligned} J(s_1) &= 1 - \varepsilon + \alpha[\varepsilon\, J(s_1) + (1-\varepsilon)\, J(s_2)], \\ J(s_2) &= \min\{\, \alpha[(1-\varepsilon)\, J(s_1) + \varepsilon\, J(s_2)], \alpha\, J(s_2)\,\}, \end{aligned}$$

with a fixed point solution equal to $J^*(s_1) = (1-\alpha)^{-1}(1-\varepsilon), J^*(s_2) = \mathbf{0}$. Therefore, the optimal stationary policy μ^* is, as it ought be, $\mu^*(s_1) = a_1$ and $\mu^*(s_2) = a_2$.

10.4.2 The QPOMDP case

For simplicity, only one observation that is always received with probability one is considered. This case corresponds to a conformant qualitative planning model. For a general kappa belief state $\kappa = [k_1, k_2]$, the transitions are given by

$$\begin{aligned} \kappa_{a_1} &= [\,\min\{1+k_1, k_2\}, \min\{k_1, 1+k_2\}\,], \\ \kappa_{a_2} &= [\,\infty, 0\,] \quad \text{(since } k_1 \text{ must be } \infty). \end{aligned}$$

Consider first the case for the simple belief states: $\kappa = [0,0]$, $\kappa' = [0,1]$ and $\kappa'' = [1,0]$. The belief transitions are then, for the only applicable control a_1: $\kappa \rightsquigarrow \kappa$, $\kappa' \rightsquigarrow \kappa''$ and $\kappa'' \rightsquigarrow \kappa'$. Therefore, the Bellman equations are:

$$\begin{aligned} J(\kappa) &= 2^{-1}(1-\varepsilon) + \alpha\, J(\kappa), \\ J(\kappa') &= (1-\varepsilon)^2 + \alpha\, J(\kappa''), \end{aligned}$$

$$J(\kappa'') \;=\; \varepsilon(1-\varepsilon) + \alpha\, J(\kappa')$$

as can be checked by computing the action costs $g(\kappa, a_1)$, $g(\kappa', a_1)$ and $g(\kappa'', a_1)$. Therefore, the fixed point solution for the basic belief states is:

$$\begin{aligned} J^*([0,0]) &= (1-\alpha)^{-1}2^{-1}(1-\varepsilon), \\ J^*([0,1]) &= (1-\alpha^2)^{-1}[(1-\varepsilon)^2 + \alpha\varepsilon(1-\varepsilon)], \\ J^*([1,0]) &= (1-\alpha^2)^{-1}[\varepsilon(1-\varepsilon) + \alpha(1-\varepsilon)^2]. \end{aligned}$$

Similarly, for the other belief states (i.e. $k > 1$), the fixed point solution is

$$\begin{aligned} J^*([0,k]) &= (1-\varepsilon^k)(1-\varepsilon) + \alpha(1-\alpha^2)^{-1}[\varepsilon(1-\varepsilon) + \alpha(1-\varepsilon)^2], \\ J^*([k,0]) &= \varepsilon^k(1-\varepsilon) + \alpha(1-\alpha^2)^{-1}[(1-\varepsilon)^2 + \alpha\varepsilon(1-\varepsilon)], \\ J^*([0,\infty]) &= (1-\varepsilon) + (1-\alpha^2)^{-1}[\varepsilon(1-\varepsilon) + \alpha(1-\varepsilon)^2], \\ J^*([\infty,0]) &= 0. \end{aligned}$$

The best stationary policy is then, $\mu^*([k_1,k_2]) = a_1$ for $k_1, k_2 \neq \infty$ since it is the only applicable action, and $\mu^*([\infty,0]) = a_2$, i.e. stay in the absorbing state s_2.

10.5 Computational Aspects

So far, the focus has been on giving sound theoretical foundations for the theory and not the computational side. However, it was shown that the value iteration algorithm can be used to find optimal stationary policies for QMDPs and QPOMDPs, and that as in the standard theory, value iteration is often a very powerful algorithm for solving general sequential decision tasks. For certain subclasses of problems however, other more specialized algorithms perform better.

Thus, for example, the class of conformant problems can be efficiently solved by doing search in the kappa belief space with standard algorithms like A* or IDA*; see Ch. 5. Such methods have proven to be powerful and successful in the standard non-deterministic and stochastic setting so they should work well in this setting.

Another important subclass of tasks is that in which all transition and observation probabilities have the same zero ranking. In such cases, the kappa belief states corresponds to sets of states and the problem reduces to the standard stochastic models with uniform probabilities which can be solved using a variety of algorithms including search.

Finally, in settings in which the number of steps is bounded a priori, the qualitative planning problem can be encoded into a (qualitative) probabilistic SAT formulas that can be solved by a qualitative version of the MAXPLAN planning algorithm of [139].

10.6 Summary and Notes

The work in kappa rankings was first formalized by Spohn [189] but its roots can be traced back to Adam's conditionals [2]. They have been used by Pearl and Goldszmidt to define a qualitative decision theory [159, 97] and are also connected with the ϵ-semantics for default reasoning [127, 157, 89]. Some of the relations between kappa rankings and probability theory are explored in [55]. More recently, Giang and Shenoy have presented a qualitative utility theory based on kappa rankings [95].

Wilson extension is an extension of the real number with infinitesimal quantities. Our extension is also another such extension. Yet, none of them is as powerful as Robinson's non-standard reals [172].

Another approach to qualitative MDPs and POMDPs has been recently proposed in terms of possibility theory [80, 178]. This approach is based on the qualitative decision criteria within the framework of possibility theory suggested in [71]. As the approach given here, their approach computes the value function and policy using a version of the value iteration algorithm. However, as their example shows, the possibilistic cost function does not have enough information to discriminate among actions, and so an optimal policy cannot be recovered from the knowledge of the optimal value function. This is a fundamental departure from the standard theory of MDPs and POMDPs in which there is a correspondence between stationary policies and cost functions.

The reason for this difference can be understood by considering the order-of-magnitude version of the Bellman equation; i.e.

$$J(s) \;=\; \min_{a \in A(s)} \; \max \{ \, g(s,a)^{\circ}, \max_{s' \in S} \; p(s,a,s')^{\circ} + J(s') \, \}.$$

It is not hard to see that this equation is usually constant for all states except the goal and hence cannot discriminate among actions. This 'loss' of information is due to the fact that order-of-magnitude quantities do not increase through summations, a fact that is well known to people in the field. This problem does not appear in the formalism presented here since full qualitative quantities instead of their order of magnitude are used to define the Bellman equations.

In summary, this chapter proposes a new formulation for a qualitative theory of sequential decision making that is based on a complete extension of the extended reals proposed by Wilson. The formal developments are achieved using ideas and methods from analysis which are mainly motivated by the necessity of taking limits. The new mathematical objects are then joined with the methods found in the standard theory of Markov Decision Processes to obtain a qualitative theory which is very close to the standard one. This is an important difference with other approaches whose ties with the standard theory of expected utility are not as clear.

Chapter 11

Extensions

In this chapter, we present to novel methods to speed up the computation of solutions for planning problems at the cost of optimality. Thus, the motivation is to tradeoff performance against optimality in a principled way.

The first section introduces the concept of on-line policies, which are policies that are computed and executed simultaneously as the agent interacts with the environment. We show how a simple modification of the HDP algorithm leads to a method for obtaining such on-line policies that achieve great savings in time and memory over some benchmark problems.

In the second section, we present a principled method for factorizing the state space into a much smaller space, and then show how a solution to such space can be 'lifted' to a solution of the original space. This novel method merges several ideas developed in the fields of model checking, knowledge compilation, and satisfiability. The benefits and limitations of the method is shown in a couple of experiments.

These two proposals are preliminary results that show promising results, and thus open ground for future research.

11.1 On-line Policies

Remember that the HDP algorithm, like FIND-and-REVISE, computes a value function J by enforcing its consistency over the states reachable from s_0 and the greedy policy. Another possible variation is to consider a similar algorithm that only enforces the consistency of the value function over the states that are reachable from s_0 and the greedy policy with some *minimum likelihood*.

For efficiency, the notion of likelihood is formalized using a non-negative integer scale, where 0 refers to a normal outcome, 1 refers to a somewhat surprising outcome, 2 to a still more surprising outcome, and so on. The measure are called *plausibilities*, although it should be kept in mind, that 0 refers to the most plausible outcomes, thus 'plausibility greater than i', means 'a plausibility smaller than or equal to i.'

The transition plausibilities $\kappa_a(s'|s)$ are obtained from the corresponding transition probabilities by the discretization:

$$\kappa(s,a,s') \overset{\text{def}}{=} \lfloor -\log_2(p(s,a,s')/\max_{s''\in S} p(s,a,s''))\rfloor$$

with $\kappa(s,a,s') = \infty$ when $p(s,a,s') = 0$. Plausibilities are thus 'normalized': the most plausible next states always have plausibility 0. These transition plausibilities are then combined by the rules of the kappa calculus [189] which is a calculus isomorphic to the probability calculus (e.g. [97]). The plausibility of a state trajectory given the initial state, is given by the sum of the transition plausibilities in the trajectory, and the plausibility of reaching a state, is given by the plausibility of the most plausible trajectory reaching the state.

The resulting algorithm, called HDP(i) for a non-negative integer i, computes a value function J by enforcing its (ϵ-)consistency over the states reachable from s_0 *with plausibility greater than or equal to* i. HDP(i) produces approximate policies fast by pruning certain paths in the search. The simplest case results from $i = 0$, as the code for

algorithm	h-track			square-2		
	time	quality	memory	time	quality	memory
HDP(h_{min})	7.547	41.894	35,835	0.275	11.130	7,245
HDP$(0,2)$	0.853	42.950	7,132	0.021	11.250	819
HDP$(0,4)$	0.826	44.000	7,034	0.022	11.500	650
HDP$(0,16)$	0.701	46.800	6,100	0.017	11.500	556
HDP$(0,64)$	0.698	46.800	5,899	0.014	11.610	564
greedy(h_{min})	N/A	47.150	356	N/A	12.450	104

algorithm	ring-3			ring-4		
	time	quality	memory	time	quality	memory
HDP(h_{min})	4.145	22.047	43,358	30.511	27.785	152,750
HDP$(0,2)$	0.311	22.000	7,145	1.758	28.500	24,195
HDP$(0,4)$	0.327	23.300	6,882	1.821	28.400	23,857
HDP$(0,16)$	0.284	23.600	6,331	1.580	30.800	21,823
HDP$(0,64)$	0.264	25.500	6,322	1.518	32.400	21,823
greedy(h_{min})	N/A	25.390	192	N/A	31.600	241

Table 11.1: Results of HDP$(0,j)$ for $j = 2,4,16,64$ and greedy policy with respect to h_{min} for $\epsilon = 10^{-3}$ and $p = 0.2$. Each value is the average over 100 executions. N/A in time for the greedy policy means 'Not Applicable' since there is no planning.

HDP(0) corresponds exactly to the code for HDP (see Alg. 13), except that the possible successors of a state s in the greedy graph are replaced by the *plausible* successors.

HDP(i) computes lower bounds that tend to be quite tight over the states that can be reached with plausibility no smaller than i. At run time, however, executions may contain 'surprising' outcomes, taking the system 'out' of this envelope, into states where the quality of the value function and its corresponding policy, are poor. To deal with those situations, a version of HDP(i), called HDP(i,j), is defined that *interleaves planning and execution* as follows. HDP(i,j) plans from $s = s_0$ by means of the HDP(i) algorithm, then executes this policy until a state trajectory with plausibility greater than or equal to j, and leading to a (non-goal) state s' is obtained. At that point, the algorithm replans from s' with HDP(i), and the same execution and replanning cycle is followed until reaching the goal. Clearly, for sufficiently large j, HDP(i,j) reduces to HDP(i), and for large i, HDP(i) reduces to HDP.

The implementation of this idea is direct from the HDP algorithm. The main difference is that in the outer-loop of the planning-and-execution algorithm, a integer representing the plausibility of the current state is kept. Thus, every time this measure goes beyond a certain threshold, a replanning step is triggered before taking the next action.

11.1.1 Experiments

This idea has been tested over the bigger racetrack problems in the benchmark used in Ch. 6. Table 11.1 shows the average cost for HDP(i,j) for $i = 0$ (i.e., most plausible transitions considered only), and several values for j (replanning thresholds). Each entry in the table correspond to an average over 100 independent executions. The average cost for the greedy policy with respect to the heuristic h_{min} is included as a bottom-line reference for the figures. Memory in the table refers to the number of evaluated states. As these results show, there is a smooth tradeoff between quality (average cost to the goal) and time (spent in initial planning and posterior replannings) as the parameter j varies. Note also that in this class of problems the h_{min} heuristic delivers a very good greedy policy. Thus, further research is necessary to assess the goodness of HDP(i,j) and the h_{min} heuristic.

11.2 Compressed Representations and Factored Spaces

The GPT system implements different heuristic search algorithms based on *explicit* descriptions of belief states, in which a belief state is represented either as a set of states or a probability distribution over states. As the numbers of states increases exponentially with the number of state variables or propositions, such explicit representations become problematic in problems with large number of relevant states within the belief states. Thus, alternate methods for representing belief states sometimes offer a solution to this 'curse of dimensionality'.

Since all the algorithms introduced so far are *independent* of the underlying representation, any improvement of the latter can be directly implemented into the search algorithms to yield better performance.

This section explores recent proposals for efficient representation of belief states, together with a novel idea to *factor* the search space in order to tradeoff solution quality against performance. These methods differ from the ones in the previous section in that they focus on how to 'change' the search space instead of changing the search algorithms.

The material is organized as follows. First, a general description of two different techniques for representing the search space using *compressed* propositional data structures are presented. These descriptions need to address two issues: how the belief states are represented and how the search space is built. Then, a method based on the second representation for factoring the search space into a potentially much smaller space is given. At the end, the ideas are illustrated with some experiments.

11.2.1 Binary Decision Diagrams

The first use of compact representation of belief states was given by Cimatti and Roveri in [51], which brought ideas from formal verification and symbolic model checking into AI planning. In their work, the authors focus on non-deterministic conformant planning problems in which the belief space is comprised of subsets of states.

In that case, each belief state b is described by a propositional logical formula φ such that

$$s \in b \iff s \models \varphi,$$

and a compact representation of φ amounts to a compact representation of the belief state b.

The same representation problem, but in a different context, appears in formal verification. In that setting, the user wants to verify that a given specification satisfies a certain condition φ; i.e. starting from a set of initial states compatible with the specification, the objective is to compute the set of reachable states given the specification and then check that all such states satisfy the condition φ [52]. Thus, when the number of reachable states is too large, an efficient representation of the set of reachable states is needed.

A general solution to this problem was given by Bryant in [42] with the introduction of a data-structure known as Ordered Binary Decision Diagram or BDD for short.

A BDD over a set of propositional symbols $\mathcal{P}$ is a directed acyclic graph such that:

(a) the graph contains one source and at most two sinks labeled TRUE and FALSE,

(b) each node in the graph, except the sink nodes, is labeled with a propositional symbol from $\mathcal{P}$,

(c) each node v, except the sinks, has two children labeled $v.T$ and $v.F$,

(d) no node has $v.T = v.F$,

(e) there exists a total order of the propositional symbols such that every path from the source to a sink respects that order, and

(f) the graph does not contain isomorphic subgraphs.

For example, Fig. 11.1 shows a BDD over the symbols $\{p_1, p_2, p_3, p_4\}$. This BDD represents the logical formula $\varphi = p_4 \wedge (\overline{p_1} \vee p_2 \vee p_3)$ since every model of the formula corresponds to a path from the source node to the sink labeled TRUE, and vice versa. For example, the model $\{p_1, p_2, \overline{p_3}, p_4\}$ generates the path $(p_1, p_2, p_4, \text{TRUE})$ while the model $\{\overline{p_1}, p_2, p_3, \overline{p_4}\}$, which does not satisfies φ, generates the path (p_1, p_4, FALSE).

In general, a BDD represents a formula φ if and only if each model of φ generates a path ending in TRUE, and each model of $\neg\varphi$ generates a path ending in FALSE.

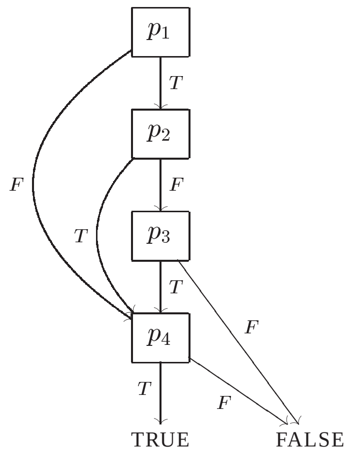


Figure 11.1: A simple BDD encoding the formula $p_4 \wedge (\overline{p_1} \vee p_2 \vee p_3)$.

What makes BDDs a suitable data-structure is that a large number of operations between BDDs can be implemented in time and space linear (or log linear) in the size of the operands. For example, all logical boolean operations can be implemented efficiently as well as the operations for model counting, testing of logical equivalence between BDDs, logical entailment between BDDs, and satisfiability of a BDD.

Two important operations that are used in the context of planning are variable substitution, and existential quantification. In variable substitution, given a BDD representing the formula φ with symbols $\{x_i\} \subseteq \mathcal{P}$ and a substitution map $\{x_i/y_i\}$ where $\{y_i\} \subseteq \mathcal{P} \setminus \{x_i\}$, a new BDD is computed representing the formula $\varphi[x_i/y_i]$ where all the x_i's have been substituted by the y_i's. In existential quantification, given a BDD representing the formula φ over symbols $\{x_i\} \subseteq \mathcal{P}$ and a symbol $x \in \{x_i\}$, a new BDD over the symbols $\{x_i\} \setminus \{x\}$ is computed representing the formula:

$$\exists x.\varphi \overset{\text{def}}{=} \varphi[x/\text{FALSE}] \vee \varphi[x/\text{TRUE}].$$

For example, if $\varphi = x \vee (y \wedge z)$, then

$$\exists y.\varphi = (x \vee (\text{FALSE} \wedge z)) \vee (x \vee (\text{TRUE} \wedge z)) \equiv x \vee (x \vee z) \equiv x \vee z.$$

The existential quantification operation is naturally extended to operate on more than one propositional symbol.

11.2.2 Search in Belief Space

The BDD operations can be used to perform search in belief space for conformant planning problems. For a set of propositional symbols $\mathcal{P}$ relevant to the planning problem, a new set of symbols $\mathcal{P}'$ is constructed with a *primed* version for each symbol in $\mathcal{P}$; i.e. $x \in \mathcal{P}$ if and only if $x' \in \mathcal{P}'$. From now on, all the BDDs are assumed to be over the set $\mathcal{P} \cup \mathcal{P}'$.

The first thing to consider is how to encode the transition relations T_a associated with action a. Let s denote a formula over $\mathcal{P}$ describing a system state, a an applicable action is s, and s' a formula over $\mathcal{P}'$ describing a possible successor state after the application of a in s, then the *propositional encoding* of the transition relation associated with a is defined as the logical formula T_a over $\mathcal{P} \cup \mathcal{P}'$ given by

$$s \wedge s' \models T_a \iff s' \in F(s,a).$$

Then, given a belief state b described by a propositional formula over $\mathcal{P}$, and an applicable action $a \in A(b)$, the belief state b_a that results from applying action a in b is given by the formula

$$b_a \equiv (\exists x.(b \wedge T_a))[x'/x] \tag{11.1}$$

where x (x') stands for all propositional symbols in $\mathcal{P}$ ($\mathcal{P}'$). The complexity of the operation described by (11.1) is only proportional to the size of b and T_a, and independent of the number of states that they represent.

The last ingredient needed, before being able to apply a standard deterministic search algorithm on belief space, is a method for checking when an action a is applicable in belief state b and whether a belief state b is a goal belief or not. In both cases, it is assumed that action preconditions and goal conditions are given by logical formulas over $\mathcal{P}$, and so both cases reduce to checking whether a belief state b entails a given formula or not. Since logical entailment is a supported operation on BDDs, such operations can be performed efficiently.

Therefore, the search space is well-defined and simple algorithms can be applied in order to find a conformant plan that achieves a goal belief state from the initial belief state.

In the case of the general non-deterministic POMDP problem, not only the transition relation is propositionally encoded but also the observation relation. In this case, it is assumed that individual observations are given by propositional formulas over $\mathcal{P}$ encoded as BDDs. Thus, if s denotes a state, a an action, and o the observation formula associated to action a, then the observation relation O_a is defined as the set of states compatible with the observation; i.e.

$$s \models O_a \iff true \in O(s,a).$$

The beliefs b_a^o that result after applying action $a \in A(b)$ in belief b and receiving observation o are then given by

$$b_a^o \equiv b_a \wedge O_a, \qquad b_a^{\overline{o}} \equiv b_a \wedge \neg O_a \tag{11.2}$$

Therefore, equations (11.1) and (11.2) define the search space in belief space for algorithms like LAO*, LRTDP, or HDPthat can be used to compute optimal solutions to the associated non-deterministic POMDP.

11.2.3 Representations using Conjunctive Normal Form

After putting in context BDDs and their use within planning in non-deterministic domains under partial information, it is now possible to describe the ideas that make up new ground to tradeoff solution quality against performance.

The first element of the approach is to switch from propositional encodings using BDDs to propositional encodings in *Conjunctive Normal Form* (CNF). As it is well known, a CNF formula is just a conjunction of clauses where each clause is a disjunction of literals, and every logical formula can be reduced to a unique CNF (modulo ordering) in which there are no *subsumed* clauses. A clause C is said to be subsumed by another clause C' iff $C' \models C$; clearly, $C \wedge C' \equiv C'$ if C' subsumes C.[1]

As before, the belief update operations are given by the same formulas (11.1) and (11.2) except it is assumed that the transition and observation relation are encoded as CNF formulas, and that the result is *normalized* back into CNF. This last normalization causes no trouble since such equations do not induce 'cross-products' over formulas.

A more difficult problem arises when checking for action preconditions or goal conditions. In this case, a full logical entailment is necessary, which is in general not computable in time proportional to the sizes of the CNF formulas involved (unless $\mathtt{p} = \mathtt{np}$).

Indeed, if b is a belief state encoded as a CNF formula, and φ is an action precondition or goal condition (also in CNF), there is no direct method of checking whether $b \models \varphi$ except by testing $b \wedge \neg\varphi$ for satisfiability; i.e. $b \models \varphi$ if and only if $b \wedge \neg\varphi$ is not satisfiable. And the latter is obviously not easy in general.[2] In this case, we propose to test for preconditions and goals with the method of *directional resolution*.

Resolution

Resolution is the standard rule of inference implemented in automatic deduction systems either for propositional or first-order theories. The completeness of resolution for refutation was given by Robinson in [173], yet see [131] for a comprehensive treatment of resolution. Here, a brief overview for propositional theories is given.

[1] From now on, clauses will be denoted interchangeably as disjunctions of literals or sets of literals.

[2] This satisfiability test is inherent to planning. Although such hard tasks are not apparent in the BDD formulation, they are hidden by the fact that building a BDD representation of the transition and observation relation is a computationally expensive operation in time and space. However, if such *compilation* is possible, then the cost can be amortized through multiple searches, or even in a single expensive search.

For two clauses $C = \{x, y_1, \ldots, y_n\}$ and $C' = \{\overline{x}, z_1, \ldots, z_m\}$ with a common propositional variable appearing positive in one clause and negative in the other, the resolution inference of $\{C, C'\}$ *upon* variable x is the clause R given by $\{y_1, \ldots, y_n, z_1, \ldots, z_m\}$. The clause R is called the *resolvent* of the clauses C and C' upon variable x, or just a resolvent. Such resolution step is denote as $\{C, C'\} \vdash R$.

A *resolution deduction* is a logical deduction in which every step is a resolution step, and a resolution schema is a method for generating resolution deductions for a given (propositional) theory. *Directional resolution* is a schema that enumerates all resolution deductions in a systematic way.

For an input set of clauses Γ and a fixed order $x_1, x_2, \ldots$ of the propositional variables, directional resolution explores the space of all deductions by sequentially generating the resolvents upon the propositional variables in the given order.

If a bucket $b[x_i]$ is assigned to each variable, an easy implementation of directional resolution can be obtained by initially partitioning the set of clauses in Γ into all those that mention x_1, those that mention x_2, and so on, assigning them to buckets $b[x_1], b[x_2], \ldots$ respectively. Then, at step i, the clauses in $b[x_i]$ are processed to generate all resolvents upon x_i which are inserted in the buckets $b[x_{i+1}], b[x_{i+2}], \ldots$ by a similar procedure. At the end, the set of all generated clauses (including Γ) is called the *directional extension* of Γ, which permits, among other things, to test for satisfiability or to generate all models of Γ in a backtrack-free manner [63]. Indeed, it can be shown that Γ is satisfiable if and only if the empty clause, denoted as $\square$, is not in its directional extension.

As it is shown in [63], directional resolution is the original Davis-Putnam resolution procedure in [58] which is different from the later and well-known unit resolution of [57].

11.2.4 Zero-Suppressed Binary Decision Diagrams

Were the CNF representation of belief states to be successful, an efficient representation of CNF formulas and implementation of resolution would be needed. Such representation is provided by the Zero-Suppressed Binary Decision Diagrams or ZBDDs for short.

The ZBDD data-structure was introduced by Minato [147] for representing general sparse sets, including sets of clauses. Here, the slightly different semantics of [46] is considered.

Plainly, a ZBDD over a set of propositional symbols $\mathcal{P}$ is an acyclic directed graph such that:

(a) the graph contains one source and at most two sinks labeled TRUE and FALSE,

(b) each node, except the sink nodes, is labeled with a *literal* from $\mathcal{P}$,

(c) each node v, except the sinks, has two children labeled $v.T$ and $v.F$,

(d) no node has $v.T = v.F$, or $v.T =$ FALSE,

(e) there exists a total order of the literals where the positive literal for a symbol is *immediately* followed by its negative version and such that every path from the source to a sink respects that order, and

(f) the graph does not contain isomorphic subgraphs.

Fig. 11.2 shows a ZBDD over the set of propositional symbols $\{p_1, p_2, p_3\}$. Similar to BDDs, the semantics is given by the collection of directed paths from the source node to the sinks. The ZBDD in Fig. 11.2, for example, represents CNF with clauses $\{p_1, p_2, \overline{p_3}\}$, $\{\overline{p_1}, p_2\}$, $\{\overline{p_1}, \overline{p_3}\}$ and $\{\overline{p_2}, \overline{p_3}\}$ since each clause represents a path from the root node to the TRUE sink while any other (non-subsumed) clause generates a path from the root to FALSE.

For a fixed ZBDD, the notation $\Delta(x, \Delta', \Delta'')$ will be used to denote the *unique* node v of the ZBDD labeled with literal x and with $v.T = \Delta'$ and $v.F = \Delta''$. Then, the set of clauses represented by node Δ, denoted $[\![\Delta]\!]$, is inductively defined as:

1. $[\![\text{FALSE}]\!] \overset{\text{def}}{=} \emptyset$,
2. $[\![\text{TRUE}]\!] \overset{\text{def}}{=} \{\square\}$, and
3. $[\![\Delta(x, \Delta', \Delta'')]\!] \overset{\text{def}}{=} (\{\{x\}\} \times [\![\Delta']\!]) \cup [\![\Delta'']\!]$;

where $\times$ and $\cup$ denote the subsumption-free cross-product and union of sets; in particular, $\mathcal{A} \times \mathcal{B} \overset{\text{def}}{=} \{A \cup B : A \in \mathcal{A}, B \in \mathcal{B}\}$. In the example of Fig. 11.2,

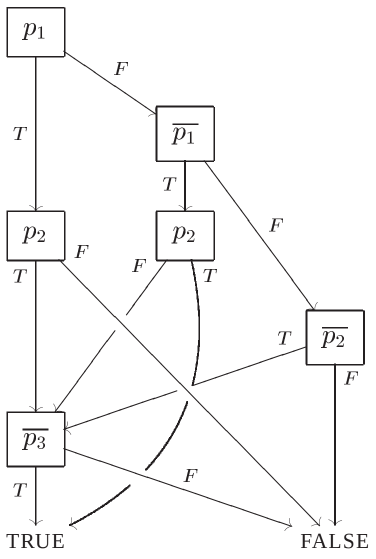


Figure 11.2: A ZBDD encoding a CNF formula with clauses $\{p_1, p_2, \overline{p_3}\}$, $\{\overline{p_1}, p_2\}$, $\{\overline{p_1}, \overline{p_3}\}$ and $\{\overline{p_2}, \overline{p_3}\}$.

$$\begin{aligned}
\Delta_1 &\overset{\text{def}}{=} [\![\Delta(\overline{p_3}, \text{TRUE}, \text{FALSE})]\!] \;=\; \{\{\overline{p_3}\}\} \times \{\Box\} \;=\; \{\{\overline{p_3}\}\}, \\
\Delta_2 &\overset{\text{def}}{=} [\![\Delta(\overline{p_2}, \Delta_1, \text{FALSE})]\!] \;=\; \{\{\overline{p_2}\}\} \times \{\{\overline{p_3}\}\} \;=\; \{\{\overline{p_2}, \overline{p_3}\}\}, \\
\Delta_3 &\overset{\text{def}}{=} [\![\Delta(p_2, \Delta_1, \text{FALSE})]\!] \;=\; \{\{p_2\}\} \times \{\{\overline{p_3}\}\} \;=\; \{\{p_2, \overline{p_3}\}\}, \\
\Delta_4 &\overset{\text{def}}{=} [\![\Delta(p_2, \text{TRUE}, \Delta_1)]\!] \;=\; (\{\{p_2\}\} \times \{\Box\}) \cup \{\{\overline{p_3}\}\} \;=\; \{\{p_2\}, \{\overline{p_3}\}\}, \\
\Delta_5 &\overset{\text{def}}{=} [\![\Delta(\overline{p_1}, \Delta_4, \Delta_2)]\!] \;=\; (\{\{\overline{p_1}\}\} \times \{\{p_2\}, \{\overline{p_3}\}\}) \cup \{\{\overline{p_2}, \overline{p_3}\}\} \\
&\qquad\qquad\qquad\qquad\quad\;\, = \{\{\overline{p_1}, p_2\}, \{\overline{p_1}, \overline{p_3}\}, \{\overline{p_2}, \overline{p_3}\}\}, \\
\Delta_6 &\overset{\text{def}}{=} [\![\Delta(p_1, \Delta_3, \Delta_5)]\!] \;=\; (\{\{p_1\}\} \times \{\{p_2, \overline{p_3}\}\}) \cup \{\{\overline{p_1}, p_2\}, \{\overline{p_1}, \overline{p_3}\}, \{\overline{p_2}, \overline{p_3}\}\} \\
&\qquad\qquad\qquad\qquad\quad\;\, = \{\{p_1, p_2, \overline{p_3}\}, \{\overline{p_1}, p_2\}, \{\overline{p_1}, \overline{p_3}\}, \{\overline{p_2}, \overline{p_3}\}\}.
\end{aligned}$$

The ZBDD data-structure supports a number of operations between sets of clauses that only depend on the sizes of the data-structures and not on the number of clauses. Among others, operations for subsumption-free union, cross-product and intersection are supported.

The main interest on ZBDDs, over other representations of clauses, is that they allow for an efficient implementation of directional resolution. Such efficiency manifests in two aspects. First, by representing set of clauses as ZBDDs and operating on them with the supported operations, no subsumed clauses are ever generated. Second, the variable elimination step on the resolution method can be implemented with ZBDD operations whose complexity only depend on the size of the data-structures.

Let Δ be a ZBDD representing a set of clauses. If Δ_x, $\Delta_{\overline{x}}$ and Δ_{out_x} stand respectively for the subset of clauses (from Δ) that contain x, contain $\overline{x}$, or do not contain either; e.g. $C \in \Delta_x$ iff $C \cup \{x\} \in \Delta$. Then, the variable elimination step, or the multi-resolution inference rule as defined in [46], is given by

$$\text{ELIMINATE}(x, \Delta) \overset{\text{def}}{=} (\Delta_x \times \Delta_{\overline{x}}) \cup \Delta_{out_x}.$$

Thus, the directional resolution of the theory Δ over variables $\{x_1, \ldots, x_n\}$ becomes simply

$$\begin{aligned}
\Delta^{(0)} &\overset{\text{def}}{=} \Delta, \\
\Delta^{(i)} &\overset{\text{def}}{=} \text{ELIMINATE}(x_i, \Delta^{(i-1)}), \qquad i = 1, \ldots, n,
\end{aligned}$$

and the directional extensions is just $\cup_i \Delta^{(i)}$. Therefore, the empty clause $\Box$ appears in some $\Delta^{(i)}$ if and only if the initial set of clauses Δ is unsatisfiable.

Armed with the benefits in representation and reasoning granted by the ZBDD data structure, the task of building a general planner based on heuristic search in belief space becomes an easy exercise. However, such a planner is only competitive with the BDD approach when the description of the planning domain, i.e. the transition and observation relations, is not efficiently compilable into BDDs.

The main motivation behind the re-formulation of the task in terms of ZBDDs is not to provide an alternative approach to planning but to make fresh ground to develop new ideas and methods. In particular, the objective now is to formulate an approach where suboptimality and solution times could be traded off in a principled and general way.

11.2.5 Factored Search Spaces

The idea of using *factored spaces* in heuristic search appears, either implicitly or explicitly, quite often. Factorizations, for example, has been used to reduce combinatorial explosion of symmetries found in problems from combinatorial optimization, planning and search.

The basic idea is quite simple: instead of considering all possible states separately, the state space is partitioned into classes of 'equivalent' states and then the smaller state space comprised of such classes is considered. In this setting, the input state space is called the base space (or just space) while the latter is called the *factored space*; the reason for this name will become apparent shortly.

The crucial concept involved here is the one of *state equivalence* since it is the relation that generates the factored space. Moreover, not every notion of equivalence is correct and, more important, the relation between the solution to the base space and the solution to the factored space depends on such concept.

Before diving into formalities, we give an illustration of these ideas is in the context of the blocks world. Consider a simple deterministic planning problem involving 26 blocks labeled with the letters A–Z in which the goal is to make the stack A on B, B on C from a known initial situation. It is obvious that in this case, all blocks not 'related' to A, B or C in the initial situation play no role in the solution and so can be safely removed from the description of the problem. Here, a block x is said to be 'related' to block y if $\texttt{above}(x, y)$ (where `above` is the transitive closure of the `on` relation)[3], and the set of blocks related to $\{A, B, C\}$ in the initial situation is the set of relevant blocks.

The removal of blocks from the initial description has a direct impact on the complexity of the search space since the branching factor in this domain is $\Omega(n^2)$ where n is the number of blocks. (Remember that we are assuming an encoding with `move` operators with two arguments.)

This example shows a simple construction of a factored space by means of the relation that considers two states (in the base space) as equivalent if and only if they agree on the positions of the relevant blocks. Clearly, a solution (optimal or not) for the factored space is a solution (optimal or not) for the base space.

Obviously, this factorization has been obtained by using knowledge specific to the blocks-world domain and it is not clear in general how to achieve such factorizations in a domain-independent way.

The rest of this chapter presents a general method for domain-independent factorizations in belief space together with its formal properties and some preliminary experimental results.

Given a belief state b represented as a set of clauses, the factorization is defined and computed in terms of its *prime implicates*.

Prime Implicates and Kernel Resolution

Given a propositional logical theory Γ, e.g. a set of clauses or more generally formulas, a *clause* C is called an implicate of Γ if $\Gamma \models C$. A prime implicate C of Γ is a *minimal implicate* in the sense that if C' is another implicate of Γ with $C' \subseteq C$ then $C = C'$. For example, the prime implicates of $\Gamma = (p_1 \vee p_2 \vee \overline{p_3}) \wedge (\overline{p_1} \vee p_2) \wedge (\overline{p_1} \vee \overline{p_2}) \wedge (\overline{p_2} \vee \overline{p_3})$ are $\{\{\overline{p_1}\}, \{\overline{p_3}\}\}$, and the set of all models for Γ can be directly obtained as they are all the assignments with $p_1 = p_3 = \text{FALSE}$ and with no restrictions on p_2 whatsoever.

In general, computing the prime implicates of a theory is as hard as satisfiability testing since the latter can be answered in linear time given the set of prime implicates. Indeed, as the example shows, not only satisfiability but all satisfying models can be computed given the prime implicates with a simple recursive procedure that never backtracks.

[3]See Ch. 4 for a formal definition of `above` in the blocks world.

A non-empty set of clauses $\mathcal{L}$ is called a *target language*, or simply language, if it is *closed under subsumption* (c.u.s.); i.e. if $C \in \mathcal{L}$ and $C' \subseteq C$ then $C' \in \mathcal{L}$. Observe that c.u.s. implies $\Box \in \mathcal{L}$ since the empty clause is always contained in any other clause. The main definition is

Definition 2 (Belief Equivalence) *Two belief states b and b' are said to be equivalent with respect to a language $\mathcal{L}$, written $b \simeq_{\mathcal{L}} b'$, if and only if $PI(b) \cap \mathcal{L} = PI(b') \cap \mathcal{L}$. In such a case, the beliefs are said to be $\mathcal{L}$-equivalent or just equivalent when the language is clear from context.*

From now on, the notation $PI_{\mathcal{L}}(\Gamma)$ will be used to denote the longer expression $PI(\Gamma) \cap \mathcal{L}$.

In the blocks-world example from above, the equivalence relation defined earlier corresponds to the relation $\simeq_{\mathcal{L}}$ where $\mathcal{L}$ is the language made of all clauses over the propositional symbols that only mention relevant blocks.

In all these definitions, the algebraic meaning of an equivalence relation is in effect; i.e. that the relation satisfies the properties of being reflexive, symmetric and transitive. Following standard algebra, any such relation partitions the space into equivalence classes whose collection forms the partition or factor space with respect to the relation (the converse also holds). Moreover, properties of the equivalence relation translate into invariants preserved between the original and factor spaces. See [87] for a gentle introduction to factor spaces from the perspective of algebra.

Before moving into the applications of the factor spaces for the general planning problem, a method known as *kernel resolution* for computing the equivalence classes (hence, the factor space) is given. In particular, the implementation based on ZBDDs from [183] is described.

Kernel resolution, given in [64], is a method for computing logical consequences of propositional theories that generalizes several resolution schemata including ordered and directional resolution as well as Tison's prime implicates algorithm [197]. Specifically, given a language $\mathcal{L}$, kernel resolution computes the set $PI_{\mathcal{L}}(\Gamma)$ for a set of clauses Γ using a variable elimination method similar to the one used for directional resolution; see [61] for an overview of general variable elimination methods in AI.

A fixed language $\mathcal{L}$ and an order $x_1, x_2, \ldots, x_n$ of the propositional symbols, which induces an order over literals, is assumed throughout.

Every clause C, either input in Γ or an intermediate resolvent, is partitioned into two parts called the $kernel(C)$ and the $skip(C)$. A *kernel resolution deduction* is a sequence of resolution steps $\{C, C'\} \vdash R$ granted by:

(a) for input clauses $C \in \Gamma$, $kernel(C) = C$ and $skip(C) = \emptyset$,

(b) resolutions are only permitted upon literals in the kernel parts,

(c) the literal l resolved upon, partitions the literals of the resolvent R into those smaller making the skip and those bigger making the kernel, and

(d) $skip(R) \in \mathcal{L}$.

It can be shown that if K denotes the set of all clauses generated in all kernel resolutions, then $PI_{\mathcal{L}}(\Gamma) = K \cap \mathcal{L}$ [64]. Thus, in order to compute the equivalence classes with respect to $\simeq_{\mathcal{L}}$ it is enough to enumerate all kernel resolution deductions in a systematic way.

To organize the enumeration of the deductions, a bucket $b[x_i]$ with clauses naming x_i is associated with each propositional symbol x_i. The clauses in each bucket are given by

$$C \in b[x_i] \iff x_i \in kernel(C) \cap \{l_1, \ldots, l_k\} \tag{11.3}$$

where $l_1, \ldots, l_k$ is the largest ordered prefix of C such that $l_1, \ldots, l_{k-1} \in \mathcal{L}$.

The systematic enumeration of the deductions is organized with a variable elimination procedure that processes all buckets $b[x_1], b[x_2], \ldots, b[x_n]$ in order, so that at step i all resolution steps over variable x_i granted by (a)–(d) are performed between the clauses in bucket $b[x_i]$, and the resolvents are then added to their corresponding buckets given by (11.3). See [64, 183] for the details and the correctness of kernel resolution.

If $\mathcal{L}$ is just $\{\Box\}$, then kernel resolution reduces to directional resolution that is equivalent to the original Davis-Putnam resolution procedure [58], while if $\mathcal{L}$ is the collection of all clauses over all propositional symbols, then kernel resolution reduces to Tison's algorithm for computing prime implicates.

```
KERNELRESOLUTION($\Gamma : clauses, k : int$)
begin
    // initialize
    $\Phi := \emptyset$;
    $\Delta := \Gamma$;

    // main loop
    for $i = n$ to $1$ do
        $\Phi :=$ [ all clauses with $k+1$ literals from $\{x_n, \ldots, x_{i+1}\}$ ];
        $\Delta := \Delta \setminus \Phi$;
        $\Delta := \Delta \cup$ ELIMINATE($x_i, \Delta$);

    // compute result
    $\Phi :=$ [ all clauses with $k+1$ literals from $\{x_n, \ldots, x_1\}$ ];
    $PI := \Delta \setminus \Phi$;
end
```

Algorithm 17: Kernel resolution algorithm of Simon and del Val that uses ZBDD data structures for computing the prime implicates over the language $\mathcal{L}_k$ on the propositional symbols $\{x_1, \ldots, x_n\}$.

Although the description of kernel resolution is somewhat complex, its implementation using ZBDDs is not, at least for the target languages considered below.

For a non-negative integer parameter k, define the target language $\mathcal{L}_k$ made of all clauses with at most k literals. Then, the equivalence class for belief state b is given by $PI_{\mathcal{L}_k}(b)$. That is, two belief states b, b' are considered equivalent by $\simeq_{\mathcal{L}_k}$ if and only if

$$PI_{\mathcal{L}_k}(b) = PI_{\mathcal{L}_k}(b'). \tag{11.4}$$

To understand this definition, remember that a state s is compatible with a belief state b if $s \models PI(b)$ and this implies $s \models PI_{\mathcal{L}_k}(b)$ since $PI_{\mathcal{L}_k}(b) \subseteq PI(b)$. Thus, the factor space with respect to $\mathcal{L}_k$ is generated by *uniformly broadening* the belief states, which is performed in *principled and systematic way* by dropping all prime implicates with more than k literals.

In the extreme cases, $\mathcal{L}_0 = \{\Box\}$, and the factor space is made of a single belief state that contains all states, while $\mathcal{L}_n$ (where n is the total number of propositional symbols) is the set of all clauses and the factor space is equal to the original belief space. In between, a complete sequence of nested spaces is obtained as the parameter k moves from 0 to n.

The kernel resolution algorithm of Simon and del Val [183] for computing the prime implicates over language $\mathcal{L}_k$ using ZBDDs is shown in Alg. 17. The algorithm is a special case (for the language $\mathcal{L}_k$) that exploits the ZBDD operations of subsumed-free set union, difference, and intersection. The algorithm is a simple variable elimination method that at each step removes all clauses containing more that k literals from the set of already eliminated variables since such clauses would generate prime implicates with more than k literals. At the end of the main loop, a final removal of clauses with more than k literals is performed. Since ZBDDs are subsumption-free representations, the ZBDD data structure for all clauses with k literals is simple and easily constructible. For example, Fig. 11.3 shows the ZBDD representing the set of all clauses with 3 literals over the set $\{x_1, \ldots, x_{m+3}\}$ of propositional symbols. Finally, the inductive variable in the main loop of the algorithm can be made to run over the set of propositional symbols in any order, yet going from x_n to x_1 allows for a more efficient implementation since each 'filter' ZBDD Φ can be constructed by extending the previous one.

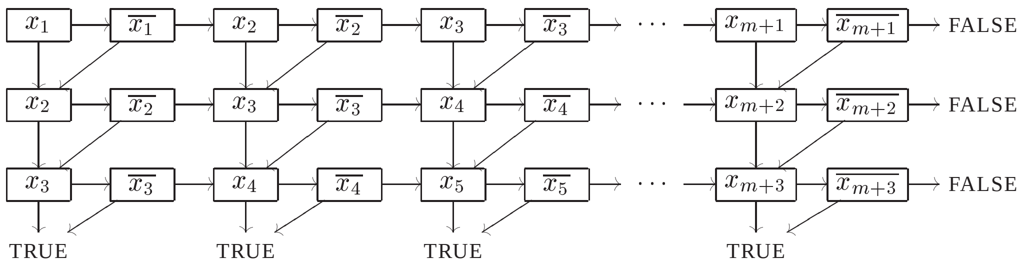


Figure 11.3: The ZBDD representing the collection of all clauses with 3 literals from the set $\{x_1, \ldots, x_{m+3}\}$ of propositional symbols. The rightwards arrows stand for the $.E$ links while the downwards for the $.T$ links.

11.2.6 A Planner based on Factored Spaces

If $[\![b]\!]$ denotes the equivalence class of the belief state b generated by (11.4), then a general non-deterministic POMDP model given by NB1–NB8 is transformed into a *factored* problem, called the k-factored version, given by:

FB1. The factored belief space $\{[\![b]\!] : b \in B\}$,

FB2. an initial belief state $[\![b_0]\!]$,

FB3. goal beliefs given by (3.12),

FB4. actions $A([\![b]\!]) \subseteq A$ applicable in each belief state $[\![b]\!]$ given by (3.11),

FB5. sets of possible observations $O([\![b]\!], a)$ given by (3.14),

FB6. a dynamics in which every action $a \in A(b)$ non-deterministically maps belief $[\![b]\!]$ into $[\![b_a^o]\!]$ for the different $o \in O(b,a)$, and

FB7. positive action costs $g([\![b]\!], a)$ given by (3.13).

The solution to this model can be obtained by the methods and algorithms given in previous chapters. For methods based on heuristic search, the only additional operation is to compute the equivalence class of each belief generated during the search, something that can be performed using the algorithm in Alg. 17. The rest is exactly the same.

The main motivation for considering the factored space is the potential reduction in size that could be achieved. In general, for a planning problem with n propositional symbols, the potential size of the belief space is $O(2^{2^n})$ since there are order 2^n possible states, and each belief state is a subset of states. For the k-factored version, however, the potential size of the belief space is $O(2^{(2n)^{k+1}})$ since there are order $(2n)^{k+1}$ clauses with at most k literals, and each belief state is a subset of such clauses. Hence, the k-factored version of the problem can be in principle significantly smaller and hence much easier to solve.

Since each belief state in the factored space is a broadened version of the beliefs in the original space, then any solution (optimal or not) to the model FB1–FB8 is a solution to the original POMDP problem NB1–NB8. Clearly, optimal solutions to the factored problem are in general not optimal for the original problem. The following result that establishes the soundness of the approach is a direct consequence of the definitions.

Theorem 60 *For a non-negative integer k, any solution to the k-factored version of the* POMDP *induces a solution to the original* POMDP*.*

The converse of this proposition is not true in general. Yet, more important is the fact that the k-factored version might not have a solution at all (in the sense of achieving the goal in a finite number of steps) while the original problem does.

The following section is devoted to characterizing the class of POMDP problems for which their k-factored versions have a solutions.

11.2.7 Bit Correlations

The only reason for which a k-factored version of a POMDP might not have a solution is due the loss of 'significant' information in the process of computing the equivalence-class representatives. Remember that computing a representative for a belief state is equivalent to dropping prime implicates with more than k literals, clauses which might contain relevant information about the models compatible with the given belief state. Thus, a factorization process involves a lost of information, and the characterization and impact of this information is the subject of this section.

First some definitions. A (non-deterministic) POMDP problem is said to be *solvable* if it admits a solution which achieves the goal in a finite number of steps.[4] A POMDP problem is said to have *strong k-bit correlations* (with respect to the goal) if the problem is solvable but its k-factored version is not, and to have *weak k-bit correlations* (with respect to the goal) if its k-factored version is solvable but its optimal solution does not induce an optimal solution for the original POMDP.

Theorem 61 *Let P be a non-deterministic* POMDP *problem. Then,*

(a) if for every solution policy π of P there exist two $\mathcal{L}_k$-equivalent beliefs b, b' in the greedy envelope such that $\pi(b) \neq \pi(b')$, then P has strong k-bit correlations, and

(b) if P is such that the actions are applicable in all states and P has weak k-bit correlations, then for every solution π of P there exist two $\mathcal{L}_k$-equivalent beliefs b, b' such that $\pi(b) \neq \pi(b')$.

Proof: In both cases, the contrapositive versions are proved. For the first claim, assume that P doesn't has strong correlations. Then, there exists a solution policy $\bar{\pi}$ for the k-factored version, which induces a policy $\pi(b) \stackrel{\text{def}}{=} \bar{\pi}([\![b]\!])$ that is a solution to the original problem (Theorem 60), and $\pi(b) = \pi(b')$ for every two equivalent beliefs.

For the second claim, assume that there exists a policy π such that every two equivalent beliefs in the greedy policy satisfy $\pi(b) = \pi(b')$. Then, the policy $\bar{\pi}([\![b]\!]) \stackrel{\text{def}}{=} \pi(b)$ is well-defined, i.e. it does not depend in the belief $b \in [\![b]\!]$, and solves the k-factored version of P. □

For problems with no (or weak) correlations, any complete planner in the factored space is complete. Here, completeness means that the planner finds a solution whenever one exists. The following is a direct consequence of the definitions and previous theorems.

Theorem 62 *If a given non-deterministic* POMDP *problem has no or weak k-bit correlations, then any solution to a k-factored version induces a solution to the problem. In the case of no correlations, an optimal solution for the k-factored version induces an optimal solution for the* POMDP*, while in the case of weak correlations the optimality of the induced solution is not guaranteed.*

11.2.8 Experiments

The approach based on factored spaces is demonstrated over two domains. In the first experiment, the planner is applied to the game of Mastermind with n pegs and two colors. Unfortunately, this instance presents strong k-bit correlations for all $k < \log n$ and thus the approach is only applicable with $k = \log n$ which generates a factor space equal to the original space. Thus, the planner based on factored spaces is not competitive with the approach based on BDDs.

The second experiment corresponds to an identification problem with n bits of information and 2^n knowledge gathering actions.

This problem was designed as a simplification of the game of Mastermind with very low correlation among the bits. The game consists of a secret code made of n bits of information that has to be discovered through guesses or tests. The test actions, that have the form of tuples $(t1, \ldots, t_n)$ of n bits, are knowledge gathering actions that only return partial information about the secret code. If we denote the secret tuple as $(s_1, \ldots, s_n)$, then upon the execution of a test action $(t_1, \ldots, t_n)$ the information returned is the logical formula:

$$(t_1 \Rightarrow s_1) \wedge (t_2 \Rightarrow s_2) \wedge \cdots \wedge (t_n \Rightarrow s_n).$$

[4] This is the same as saying that there exists a proper policy for the POMDP; see Ch. 3.

```
types     = obj bool
relations = on(?obj)
            value(?bool)
objects   = s1 s2 s3 s4 s5 - obj
            T F - bool

test(?t1,?t2,?t3,?t4,?t5)
    types: ?t1 ?t2 ?t3 ?t4 ?t5 - bool
      obs: ((¬value(?t1) ∨ on(s1)) ∧
            (¬value(?t2) ∨ on(s2)) ∧
            (¬value(?t3) ∨ on(s3)) ∧
            (¬value(?t4) ∨ on(s4)) ∧
            (¬value(?t5) ∨ on(s5)))

init: value(T) ∧ ¬value(F)
goal: full-knowledge
```

Figure 11.4: Description of the identification problem for $n = 5$.

For example, if $n = 3$, the secret code is $(true, false, true)$ and the test action is $(false, false, true)$, then the information returned by the action is

$$(false \Rightarrow true) \;\wedge\; (false \Rightarrow false) \wedge\; (true \Rightarrow true) \;\equiv\; true\,,$$

while if the test action is $(true, true, true)$, then the feedback is

$$(true \Rightarrow true) \;\wedge\; (true \Rightarrow false) \wedge\; (true \Rightarrow true) \;\equiv\; false\,.$$

A description of this game in the representation language for $n = 5$ is shown in Fig. 11.4. As shown, the possible initial situation is given by the set of 2^5 combinations of the information bits plus a statement that assign the truth values $true$ and $false$ to the constants T and F.

This problem has no correlations among the bits, so an optimal solution to the factored problem generates an optimal solution to the original problem.

The solution times and memory consumption of the planner based on kernel resolution, for different values of k, are shown in Table 11.2. Two planners are considered, a base planner that uses BDDs for representing belief states, and the factored planner that uses ZBDDs, and kernel resolution for factoring the belief space. As can be seen, the original belief space grows very fast making the planner based on BDDs impractial for relatively small problems while the factored space grows more slowly. However, $k = 1$ seems to be the only possible choice for bigger problems.

The main lesson to draw from these experiments is that the idea of searching factored spaces might be a solution to the combinatorial explosion in belief space in some domains and so deserves more attention. The approach presented here shows one such factorization but other approaches are also possible. The factorization of search spaces is currently an active field of research.

11.3 Summary and Notes

Preliminary work on two methods for trading off solution quality against time has been presented. The first method is based on a version of the HDP algorithm that interleaves planning and execution, hence delivering non-optimal solutions. The basic idea is to find a solution only for the most plausible states and then, to launch replanning steps (during execution) whenever a transition is made to a state not previously considered. The plausibilities of the transitions are measured using the kappa calculus which is a calculus isomorphic to probabilities. Kappa calculus has been used in a variety of scenarios including knowledge representation and default reasoning, and in Ch. 10 to define qualitative versions of the different planning tasks.

			ZBDDS					
prob.	feature	BDDS	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$
$n = 2$	time	0.07	0.07	0.08				
	memory	10	8	10				
$n = 3$	time	0.13	0.08	0.12	0.12			
	memory	37	22	37	37			
$n = 4$	time	1.50	0.33	1.36	2.05	2.04		
	memory	273	68	214	273	273		
$n = 5$	time	106.60	1.92	26.60	121.28	146.03	145.05	
	memory	6,589	197	1,693	5,899	6,571	6,589	
$n = 6$	time	$>$ 2h	13.14	631.62	$>$ 2h	N/A	N/A	N/A
	memory		626	13,940				
$n = 7$	time	N/A	102.57	$>$ 2h	N/A	N/A	N/A	N/A
	memory		2,009					

Table 11.2: Results in time and memory for different instances of the identification problem for the planner working in the plain belief space (BDD), and the planner working in the factored belief space (ZBDD) for different values of k. N/A means not attempted.

These ideas are then tested using some benchmark problems. The results show that the approach is worth exploring more deeply.

Another way of understanding the on-line planner is in terms of envelopes. Thus, in this case the first planning step computes a policy for an initial envelope of most plausible states and then refines the envelope when unlikely states are visited. The use of explicit envelopes that are gradually extended is also present in [59] and [192]. Interestingly, these envelopes are expanded by including the most likely reachable states not yet in the envelope

The second method is based on factoring the search space into classes of equivalent states. The factorization is defined in terms of a propositional representation of the belief states based on prime implicates. Similar techniques used to represent belief states were first used in model checking and knowledge compilation; see [52] for an overview of model checking, and [181, 56] and references therein for an overview of concepts and techniques of knowledge compilation.

For the second method, necessary and sufficient conditions for the soundness and completeness of the approach are given. The benefits and limitations of the second approach are also explored in a number of experiments with interesting results.

Other related work include the use of factored representations for value functions, and algorithms based on such representations. These approaches are based on more general binary decision diagrams known as ADDs whose sink nodes are permitted to be arbitrary reals and not only boolean values. Thus, ADDs can be used to represent general functions. Symbolic versions of value iteration and other algorithms have been given in terms of these data structures, and shown to be successful in a number of well-structured problems; see [105, 39, 81].

Propositional representations are only directly applicable to propositional planning problems and not to problems expressed in the f-STRIPS language. However, any given instance of a problem expressed in f-STRIPS can be translated into a propositional encoding by treating the terms as different propositional symbols. Such encoding is however exponential in the nesting-level of the terms and the number of objects.

The removal of redundant propositional symbols as in the world-blocks example is similar to the removal of static atoms and logical invariants in classical planners; e.g. HSP [30] and STAN [85].

Chapter 12

Directions for Future Research

Different directions for current and future work are given in this chapter. The topics are classified, following the basic approach, into mathematical models, description languages, and algorithms.

12.1 Mathematical Models

The underlying mathematical models for the class of planning problems dealt with in this dissertation are clear and well understood. A topic of current research is to extend the basic setting to allow more general time and state-space settings, and more complex actions; e.g. infinite state spaces and durative actions. Relevant mathematical models in this regard are semi-Markov decision processes, Markov processes in continuous time, and general diffusion processes. Although a very general formulation could be obtained using a general Markov process in continuous time, it is preferable to obtain restricted classes of mathematical models for some classes of problems.

A more interesting area of research is the study of different optimization criteria. As was seen, the infinite-horizon expected discounted cost is a very simple setting with nice clean results. The undiscounted case is a hot area of research where the main open problems (for years now) are that of bounding the speed of convergence and suboptimality of greedy policies. Other directions of research for undiscounted models involve showing the existence and uniqueness of solutions for Bellman's equations under weaker conditions than the ones assumed here.

Throughout this dissertation, the optimization criteria has been set to minimizing the expected cost. Another possibility that is frequently used is minimizing the (discounted or undiscounted) average cost per step; see [164].

As was seen in Ch. 3, the study of different stochastic processes can provide results on the speed of convergence for value iteration and algorithms like RTDP. This kind of argument based on couplings of processes seems to be promising for the analysis of the different mathematical models.

For the case of POMDP models little is known. Stronger properties about the optimal value function would automatically translate into better algorithms; e.g. a tight bound on the modulus of continuity of the optimal value function can be used to induce better discretizations. Similarly, stronger results about Sondik's representation theorem could translate directly into better algorithms.

12.2 Description Languages and Representations

There is a growing interest in the planning community towards achieving a consensus on a standard description language for planning under uncertainty and partial information. This work presents a first proposal in that direction that is an extension of the PDDL language for deterministic planning (which is currently the accepted standard).

The proposed language in Ch. 4 is a simple action language aimed at descriptions at the level of states and with almost no support for expressing *epistemic* constructs, the sole exception being the condition `full-knowledge` for goal states. A simple and important generalization is to extend the language to treat such modalities as first-class expression allowing for a more expressive language. Thus, for example, the action ‘call Jenny at her cell phone’ would require knowing Jenny’s cell-phone number which could be obtained by consulting the phone book.

Another interesting possibility in this direction is to move to languages aimed at descriptions at the level of beliefs. In such a case, the logics involved are termed epistemic or belief logics which have received a lot of attention in AI; see, for example, [79]. It is known that some of these logics can be naturally translated into propositional form using three-valued logics and thus are amenable to efficient implementations based on compressed propositional structures as the ones found in formal verification and model checking. See Ch. 11 for an introduction to such structures.

Obviously, new languages and representation techniques need new algorithms and heuristic functions to capitalize on them.

12.3 Algorithms and Heuristics

The necessity of new and better algorithms is always there, as well as better heuristic functions to seed the algorithms. Especially relevant is work directed towards important subclasses of problems. For example, [146] recently introduced the concept of limited-contingency plans and a first proposal for their computation.

The case of heuristic functions is a hot topic of research since they are responsible for the success of the heuristic search approach to general planning. The current heuristic functions are good enough to solve interesting but rather small problem instances. For scaling up, novel and more powerful heuristic functions are needed. Furthermore, if the interest is in domain-independent methods, heuristic functions computed from problem descriptions are needed. A concrete open problem in this regard is how to compute informative heuristic estimates for problems described using the functional capabilities of the description language in Ch. 4.

Another important area of research is approximate on-line or off-line algorithms. As was shown in Ch. 11, on-line algorithms might offer practical solutions to a number of real-world problems where the main interest is not in finding optimal policies but just good policies. A good policy could be defined as a proper policy (i.e. one that eventually achieves the goal), or an almost proper policy (i.e. one that with probability $1 - \delta$ eventually achieves the goal).

Finally, approximation algorithms working in factorizations of the state space (as defined in Ch. 11) could potentially achieve improvements in performance of several orders of magnitude. These factorizations can be used for different purposes such as removing symmetries in the search space, exploiting weak correlations among belief states, etc.

Chapter 13

Conclusions and Contributions

In this dissertation, the general task of sequential decision making under uncertainty and partial information is cast as heuristic search in belief space. For the success of this approach three different aspects should be taken into account: the mathematical models that formalize the different tasks and their solutions, the representation language for describing the models implicitly, and the solution algorithms.

Besides the general contribution of casting sequential decision tasks into search problems, this dissertation provides specific contributions for three aspects. For the mathematical models, new theoretical results about the existence of solutions, and the convergence of known and new algorithms are shown. These results either generalize previous results, offer new simpler proofs, and/or show new properties about specific models.

With respect to representation languages, a novel and very expressive logic-based language is proposed. The language can be understood as an extension of the STRIPS language widely used in AI that allows for expressing non-determinism and feedback signals. The expressiveness of the language is demonstrated with a number of problems which includes the PSR domain (in Ch. 9) which, up to the date of writing, is not expressible in other representation languages. The semantics of the language is given in terms of the mathematical models instead of using first-order logic as in other approaches. This semantic approach makes it possible to provide syntactic constructs which are very difficult or impossible to formalize using FOL; e.g. transitive closure.

As for the algorithms, this dissertation introduces a new general family of algorithms, called FIND-and-REVISE, that exploit known lower bounds (heuristic functions) and the knowledge of the initial state (belief) in order to compute partial optimal policies that are closed with respect to the initial state. As shown in the experiments, the new algorithms outperform previously known algorithms, like dynamic programming, and can be used to solve problems beyond the reach of current algorithms.

Perhaps the most general and relevant contribution of this dissertation is to make an explicit and useful relation between the traditional search models of AI and the sequential decision models of control theory. For example, the traditional solution methods in control theory are based on explicit models, and do not use any information about the initial state of the system and known lower bounds. Indeed, standard reference books in control theory like [16] and [164] do not mention heuristic functions or initial states, nor their role when computing solutions. On the other hand, search methods found in AI are aimed towards implicit models, and use heuristic functions and knowledge about the initial state to prune the combinatorial explosion during the search, yet such methods were not applied to the kinds of mathematical models dealt with in this dissertation.

Thus, the relation between both approaches given here opens the door for a fruitful cross fertilization among the two disciplines that will lead to the development of better algorithms and methods capable of pushing forward the frontier of tractable problems.

Other interesting proposals are the extensions of the standard model presented in chapters 10 and 11 which include order-of-magnitude approximations for sequential decision tasks only known up to some degree of precision, on-line algorithms that can be used to interleave planning and execution, and general techniques based on propositional logic that can are used to factorize the search space. All these extensions are aimed at the goal of trading off solution quality against computation time in a principled way.

Most of the ideas contained in this dissertation have been implemented into a general planning system, and tested on diverse problems like robot navigation, medical diagnosis, synthesis of circuits, game playing, power restoration, etc. The system and benchmark problems are all available to the general public in order to expedite the dissemination of the ideas contained here.

13.1 Specific Contributions

We list, by chapter, the most important specific contributions of this dissertation.

Chapter 2. All planning models considered in AI are cast in terms of general state models with different transition functions and sensor models. The relation between these models and the mathematicals models is given.

Chapter 3. We give formal definitions of non-deterministic MDP and POMDP models in this chapter plus a characterization of their optimal solution based on Bellman's equations. Although many researchers are aware of this type of characterization, no formal definitions or proofs are found in the literature to my knowledge. Thus, Theorem 1 and its proof are important contributions.

For stochastic shortest-path problems, Theorem 6 is an important generalization of the result found in [16] to the case of infinite state spaces; its proof, an argument based on compactness, is a novel one. We also provide a bound on the convergence rate for SSP problems in Theorem 7 and Theorem 11.

For POMDP models, Theorems 13, 14, and 15 define a new metric space, show that the optimal value function is a uniformly continuous functional defined over such a space, and give a bound on its modulus of continuity. These are very important contributions that allow the development of optimal algorithms for POMDPs based on discretizations of the belief space. We also show that when all action sets are equal, then the optimal value function is uniformly continuous with respect to the supremum norm.

Finally, for the case of SSP problems in belief space, we give a characterization of optimality under some conditions in Theorem 18. This is also an important contribution.

Chapter 4. We provide a representation language for general planning tasks that is based on the STRIPS language. The language supports syntactic constructs for expressing non-determinism and observation effect. The language also supports simple recursive definitions which are interpreted using the stratified models of logic programs. This is one of the first action languages with such broad scope.

Chapter 6. This chapter deals with algorithms for MDP models. The most important contributions are the idea of computing partial optimal policies closed with respect to the initial state, and the development of the algorithms LRTDP, FIND-and-REVISE and HDP. We provide proofs of correctness and complexity of all these algorithms in Theorems 31, 32, 35, 36 and 37.

We also provide a novel proof for the convergence of the RTDP algorithm in Theorem 25. A bound for the expected termination time of each RTDP trial is given in Theorem 26; its proof uses an argument similar to the one of Theorem 11.

Theorem 22 shows that a small residual is enough to guarantee an optimal greedy policy for the case of SSP problems. This is an important result that motivates the interest on algorithms that compute value functions with small residual. (I have not seen a result like this published elsewhere but it seems that some researches are aware of it.)

Another interesting contribution is the empirical evaluation of the value iteration algorithm and LAO* against the novel algorithms LRTDP and HDP.

Chapter 7. We provide new grid-based algorithms for POMDP models. Using the result from Ch. 3, we develop an optimal and complete grid-based algorithm for POMDPs. This is the first grid-based algorithm for POMDPs that is able to offer optimality guarantees of the resulting greedy policy as shown in Theorem 39 and Corollary 40.

The LRTDP and FIND-and-REVISE algorithms are casted into belief space to obtain other algorithms for POMDP models. As shown in the experiments, these algorithms outperform previously known algorithms by orders of magnitude on benchmark problems.

Another contribution of this chapter is the definition of a novel criterion of robustness for algorithms based on discretizations. As shown, all novel algorithms are robust in such sense while algorithms based on plain discretizations are not.

Chapter 8. New domain-independent heuristic functions for the different mathematical models are given in this chapter.

For MDP models, the h_{min} and h^k_{min} heuristic functions are given together with their admissible and monotonic properties in Theorem 43.

For POMDP models, the h_{mdp}, h^k_{mdp}, h_{sup} and h^m_{sup} heuristic functions are defined, and their admissible and monotonic properties are established in Corollary 46 and Theorems 47, 48, 49 and 50.

For the case of DDP models, we show that the heuristic h^k_{max} of [102] is admissible and monotonic.

Chapter 9. Several experiments that show the capabilities of the GPT planner are given in this chapter. For the most complex domain of PSR, we give a formal encoding of it in the representation language. Up to my knowledge, this is the first formal encoding of the PSR domain that is independent of the network topology. Previous encodings of PSR were based either in propositional or first-order logic which cannot express connectivity relations in general network topologies.

The experimental results show that GPT is able to compute optimal policies for problems that are beyond reach of other planners.

Chapter 10. This chapter presents a qualitative theory for general sequential decision making based on kappa functions. This chapter is an extension of [34]. All results from Lemma 52 to Corollary 59 are new and original contributions.

Chapter 11. Two methods for trading off solution quality against performance are given in this chapter. The first method is a novel contribution that shows how to interleave planning and execution in a principled way within a reactive system.

The second method introduces a novel idea of how to factorize the search space into a simpler space. The factorization is defined in terms of prime implicates and kernel resolution, and its granularity is controlled with an integer parameter k. The soundness of the factorization is shown in Theorem 60, while a characterization of the class of problems that admit solvable factorizations is given in Theorems 61 and 62.

Appendix A

Brief Mathematics Review

This appendix contains standard definitions and results from analysis and probability that can be found in any good reference. Highly recommended are [174, 175] for general analysis, and [88, 72, 19] for probability theory. This appendix is inspired in format and style by the one in [19].

Analysis

A1. *O Notation.* A function f is said to be $O(g)$, read 'big-oh' of g and written $f = O(g)$, if there exists a positive constant K and integer N such that $|f(n)| \leq K|g(n)|$ for all $n > N$. It is said to be $\Omega(g)$ if $g = O(f)$, and $\Theta(g)$ if it is $O(g)$ and $\Omega(g)$. A function f is said to be $o(g)$, read 'little-oh' of g, if $|f(n)|/|g(n)| \to 0$ as $n \to \infty$. The 'big-oh' notation is due originally to [8] while the 'little-oh' to [130].

A2. *Least Upper Bound Property.* Any set X of real numbers bounded by above has a least upper bound denoted by $\sup X$, i.e. if α is any other upper bound then $\sup X \leq \alpha$. Similarly, any set bounded from below has a greatest lower bound denoted by $\inf X$.

A3. *Monotonic Sequences.* The sequence $\{a_n\}_n$ is said to be monotonic non-decreasing (non-increasing) if $a_n \leq a_{n+1}$ ($a_n \geq a_{n+1}$). In a similar way, increasing and decreasing sequences are defined.

A monotonic sequence is either bounded or unbounded. In the latter case, the sequence is said to diverge to either ∞ or $-\infty$ depending if it is non-decreasing or non-increasing. In the former case, there exists a number a such that for $\epsilon > 0$ there exists N with $|a - a_n| \leq \epsilon$ for all $n > N$. In this case, a is said to be the limit of the sequence written $\lim_{n\to\infty} a_n = a$, $a_n \to a$ as $n \to \infty$, or simply $\lim a_n = a$ and $a_n \to a$ when $n \to \infty$ is clear from context.

Similar definitions apply for functions (or vector); e.g. a sequence of functions $\{f_n\}_n$ defined over X is non-decreasing if each sequence $\{f_n(x)\}_n$ for $x \in X$ is non-decreasing. In this case, the pointwise limit of the sequence of functions is defined to be the function f if $\lim f_n(x) = f(x)$ for all $x \in X$. Below, other kinds of convergence are considered.

A4. *Limits.* For general sequences $\{a_n\}_n$, convergence to a real a is defined if for all $\epsilon > 0$ there exists N such that $|a_n - a| \leq \epsilon$ for all $n > N$. Similarly, $\{a_n\}_n$ has as a limit point the real a if for all $\epsilon > 0$ and integer N, there exists $n > N$ such that $|a - a_n| \leq \epsilon$. Clearly, $\lim a_n = a$ iff a is the unique limit point of $\{a_n\}_n$.

The inferior limit of $\{a_n\}_n$ is defined to be the limit of the monotonic sequence $\{b_n\}_n$ where $b_n = \inf\{a_m : m \geq n\}$ and, by [A3], the inferior limit always exists. The inferior limit is denoted by $\liminf a_n$. Similarly, the superior limit, denoted by $\limsup a_n$, is defined by considering $\sup\{a_m : m \geq n\}$. Obviously, $\liminf a_n \leq \limsup a_n$, and the relation between the different limits is given by $\lim a_n = a$ iff $\liminf a_n = \limsup a_n = a$.

Similar definitions apply for functions, see [A3]. Thus, f is said to be the pointwise limit of $\{f_n\}$ if $\lim f_n(x) = f(x)$ for all x in the domain of f_n.

A5. *Metrics.* A metric σ over a set X is a function $\Omega \times \Omega \to \mathbb{R}^+$ such that (i) $\sigma(x,y) \leq 0$ for all $x, y \in X$ and $\sigma(x,y) = 0$ iff $x = y$, (ii) $\sigma(x,y) = \sigma(y,x)$, and (iii) $\sigma(x,y) \leq \sigma(x,z) + \sigma(z,y)$. This last inequality is known as the triangle inequality.

A pair (X, σ) where σ is a metric over X is said to be a metric space. When the metric is clear from context, then is standard to denote the metric space by just X.

A6. *Balls, Closed and Open Sets.* The set of points $B(x,r) \overset{\text{def}}{=} \{y : \sigma(x,y) < r\}$ for $x \in X$ and positive r is the *open* ball centered at x with radius r. If the strict inequality is replaced by the weaker $\leq$, the ball is closed.

More generally, a subset $O \subseteq X$ is said to be open, if for all $x \in O$ there exists $r > 0$ such that $B(x,r) \subseteq X$. The complement of an open set is a closed set. Every set A has a biggest open set contained in it, called the interior of A, and a smallest closed set containing it, called the closure of A. The boundary is defined as the difference between the closure and the interior. A set D is dense in X if its closure is equal to X.

A7. *Compact Sets.* A compact subset of $\mathbb{R}$ (or $\mathbb{R}^n$) is a closed and bounded subset. Every sequence of elements inside a compact set is bounded and has at least one limit point inside the compact. This is the Bolzano-Weierstrass theorem.

More generally, a subset X of a metric space is compact if every cover of X with open sets $\{O_k\}$ has a finite subcover of X, i.e. there is N such that $X \subseteq \cup_{0 \leq k \leq N} O_k$. In the case of real numbers, this is the Heine-Borel theorem which implies Bolzano-Weierstrass.

A8. *Pointwise Convergence* A sequence of functions $\{f_n\}$ with domain X is said to converge pointwise to a function f (with domain X) if $\lim f_n(x) = f(x)$ for all $x \in X$.

A9. *Complete and Separable Metric Spaces.* A sequence $\{x_n\}$ from a metric space (X, σ) is said to be fundamental if $\sigma(x_n, x_m) \to 0$ as $n, m \to \infty$. $\{x_n\}$ has limit $x \in X$ iff for all $\epsilon > 0$ there exists N such that $\sigma(x, x_n) < \epsilon$ for all $n > N$.

A metric space X is complete if every fundamental sequence has a limit in X. A metric space X is separable if it has a countable dense subset.

A10. *Totally Bounded Metric Spaces.* An ϵ-net for a metric space (X, σ) is a set $Z \subseteq X$ such that $x \in X$ implies there is $z \in Z$ with $\sigma(x,z) \leq \epsilon$.

The metric space X is totally bounded if there is a finite ϵ-net for each $\epsilon > 0$.

A11. *Vector Spaces.* A set of functions V over domain X is a real vector space if it is closed under (pointwise) addition of functions and (pointwise) product of reals (scalars) times functions. If X is a finite set with n elements, the space is n-dimensional, otherwise it may be infinite dimensional.

A12. *Norms and Banach Spaces.* A norm $\|\cdot\|$ over a vector space is a non-negative valued function such that (i) $\|J\| \geq 0$ for all J and $\|J\| = 0$ iff $J = 0$, (ii) $\|\alpha J\| = |\alpha| \, \|J\|$, and (iii) $\|J + J'\| \leq \|J\| + \|J'\|$ for all J, J'. A pair $(V, \|\cdot\|)$ is a normed vector space. Any norm induces a metric $\sigma(x,y) \overset{\text{def}}{=} \|x - y\|$. If the vector space is complete as a metric space, then it is a Banach space.

Standard norms are the sup or max norm $\|J\| \overset{\text{def}}{=} \sup_{x \in X} |J(x)|$, the L_1 norm $\|J\|_1 \overset{\text{def}}{=} \sum_{x \in X} |J(x)|$, the L_2 norm $\|J\|_2 \overset{\text{def}}{=} \{\sum_{x \in X} J(x)^2\}^{1/2}$, and (in general) the L_p norm $\|J\|_p \overset{\text{def}}{=} \{\sum_{x \in X} J(x)^p\}^{1/p}$.

A13. *Uniform Convergence.* A sequence of functions $\{f_n\}$ is said to converge uniformly to f if $\{f_n\}$ converges to f in the metric space induced by the sup norm; i.e. if $\|f_n - f\| \to 0$ as $n \to \infty$. More generally, the sequence is said to converge in L_p if the sequence converges in the metric space induced by the L_p norm.

A14. *Contractions.* An operator T defined on a closed subset S of a normed vector space over X, with norm $\|\cdot\|$, is said to be a contraction over S if (i) S is stable under T, i.e. $TJ \in S$ whenever $J \in S$, and (ii) there exists $\alpha < 1$ such that $\|TJ - TJ'\| \leq \alpha\|J - J'\|$ for all $J, J' \in S$. Any such contraction has a unique fixed point $J^* \in S$ which satisfies $T^k J \to J^*$ as $k \to \infty$ for all vectors $J \in S$. Additionally, T is a continuous operator [A17] with $TJ_n \to TJ$ whenever $J_n \to J$ for $J_n, J \in S$.

A15. *Pseudo Contractions.* An operator T defined on a closed subset S of a normed vector space over X, with norm $\|\cdot\|$, is said to be a pseudo-contraction over S if (i) S is stable under T, i.e. $TJ \in S$ whenever $J \in S$, (ii) there exists a fixed point $J^* \in S$ of T, and (iii) there exists $\alpha < 1$ such that $\|J^* - TJ\| \leq \alpha\|J^* - TJ\|$ for all $J \in S$. In such case, $T^k J \to J^*$ for all $J \in S$.

A16. *Topology.* A topology for set X is a collection $\mathcal{O}$ of subsets of X, called the open sets, such that (i) $\emptyset \in \mathcal{O}$, (ii) $X \in \mathcal{O}$, and (iii) $\mathcal{O}$ is closed under arbitrary unions, i.e. $\cup_{\alpha \in I} O_\alpha \in \mathcal{O}$ for all index set I with $O_\alpha \in \mathcal{O}$ for $\alpha \in I$. Any metric σ over X generates a topology $\mathcal{O}$ given by all open sets with respect to σ.

A17. *Continuous Functions.* A real-valued function f over a metric space X is continuous at $x \in X$ if for all $\epsilon > 0$ there exists $\delta > 0$ so that $|f(x) - f(y)| \leq \epsilon$ whenever $\sigma(x, y) \leq \delta$. More generally, a function $f : X \to X'$ between two metric spaces (X, σ) and (X', σ') is continuous at $x \in X$ if for all $\epsilon > 0$ there exists $\delta > 0$ so that $\sigma'(f(x), f(y)) \leq \epsilon$ whenever $\sigma(x, y) \leq \delta$. The function is continuous over X if it is continuous at each $x \in X$. The function is uniformly continuous over X if the δ in the definition doesn't depend on x, i.e. for all $\epsilon > 0$ there exists $\delta > 0$ so that $\sigma'(f(x), f(y)) \leq \epsilon$ whenever $\sigma(x, y) \leq \delta$.

From a topological perspective, a function $f : X \to X'$ between two topological spaces $(X, \mathcal{O})$ and $(X', \mathcal{O}')$ is continuous at $x \in X$, if $f^{-1}(O') \in \mathcal{O}$ for all $O' \in \mathcal{O}'$ containing x. Thus, a function $f : X \to X'$ is continuous with respect to the metric spaces (X, σ) and (X', σ') iff it is continuous with respect to the topologies induced by σ and σ'.

A18. *Inner Products.* A inner (or scalar) product over a vector space V is a real-valued function with domain $V \times V$, denoted (f, g) for $f, g \in V$, such that (i) $(f, f) \geq 0$ and $(f, f) = 0$ iff $f = 0$, (ii) $(f, g) = (g, f)$, (iii) $(\alpha f, g) = \alpha(f, g)$ and (iii) $(f, g + h) = (f, g) + (f, h)$ for all $f, g, h \in V$ and $\alpha \in \mathbb{R}$.

A inner product induces a norm by $\|f\| \overset{\text{def}}{=} (f, f)$ which in turn induces a metric over V [A12]. If the inner-product space is complete as a metric space it is called a Hilbert space.

Probability

A19. *Fields and σ-Fields.* Fix a set Ω of possible *outcomes* and denote with A^c the complement with respect to Ω of a subset $A \subseteq \Omega$; i.e. $A^c = \Omega \setminus A$. A collection $\mathcal{F}$ of subsets of Ω is a σ-field if (i) $\emptyset \in \mathcal{F}$, (ii) $A^c \in \mathcal{F}$ whenever $A \in \mathcal{F}$, and (iii) $\mathcal{F}$ is closed under countable unions, i.e. $\cup_n A_n \in \mathcal{F}$ for all $\{A_n\} \subseteq \mathcal{F}$. If, instead of (iii) the weaker condition that $\mathcal{F}$ is closed under finite unions holds, then $\mathcal{F}$ is a field.

A20. *Measure and Probability Spaces.* A measurable space is a pair $(\Omega, \mathcal{F})$ where Ω is a set and $\mathcal{F}$ is a σ-field of subsets of Ω. A measure over a measurable space is a non-negative real-valued set function $\mu : \mathcal{F} \to \mathbb{R}^+$ such that (i) $\mu(A) \geq 0$ for all $A \in \mathcal{F}$, (ii) $\mu(\emptyset) = 0$, and (iii) $\mu(\cup_n A_n) = \sum_n \mu(A_n)$ for every countable collection $\{A_n\}$ of disjoint subsets from $\mathcal{F}$. A measure space is a tuple $(\Omega, \mathcal{F}, \mu)$ so that the first two elements conform a measurable space and μ is a measure over it. A probability space is a measure space with $\mu(\Omega) = 1$; in this case, any set $A \in \mathcal{F}$ is termed as an *event*.

A21. *Almost Sure Events.* An event $A \in \mathcal{F}$ is called an almost sure event if $\mu(A) = 1$. Clearly, Ω is an almost sure event, A is almost sure iff $\mu(A^c) = 0$, and the intersection of a countable number of almost sure events is an almost sure event (exercise).

A22. *Cylinder Sets.* Quite often, e.g. as found in Markov processes, probability spaces are induced from outcomes that belong to (infinite) product spaces; i.e. $\Omega \subseteq X^\infty$. In this case, the natural σ-algebra to consider is the *smallest* σ-algebra that contains all cylinder sets. The cylinder set A generated by the collection of sets $\{A_i : 1 \le i \le n, A_i \subseteq X\}$ is defined as $\{(x_1, x_2, \dots) : x_i \in A_i, 1 \le i \le n\}$.

Clearly, the infinite product space X^∞ can be replaced by the more general product space $\Pi_i X_i$.

A23. *Measurable Functions and Random Variables.* A function $f : (\Omega, \mathcal{F}) \to (\Psi, \mathcal{G})$ is said to be a $\mathcal{F}/\mathcal{G}$-measurable iff $f^{-1}(A) \in \mathcal{F}$ for all $A \in \mathcal{G}$. If the function f is real valued, it is standard to regard $\mathcal{G}$ as the smallest σ-field containing all open sets, called the Borel σ-field, and typically denoted with $\mathcal{B}$.

In the case of probability spaces, a real-valued random variable is a $\mathcal{F}/\mathcal{B}$ measurable function, and a real-valued random vector is a $\mathcal{F}/\mathcal{B}^n$ measurable function where $\mathcal{B}^n$ denotes the Borel σ-field for $\mathcal{R}^n$; i.e. the smallest σ-field containing all open sets in the n-dimensional Euclidean space.

Appendix B

Action Language Abstract Grammar

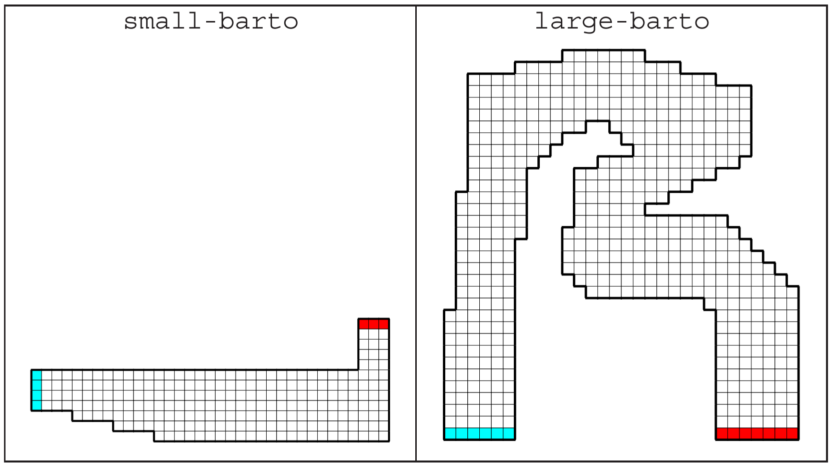
As it is usual, non-terminal symbols are denoted between angle brackets, reserved words with lowercase, and proper tokens in capitals. The symbol ϵ denotes the empty string. The abstract grammar is:

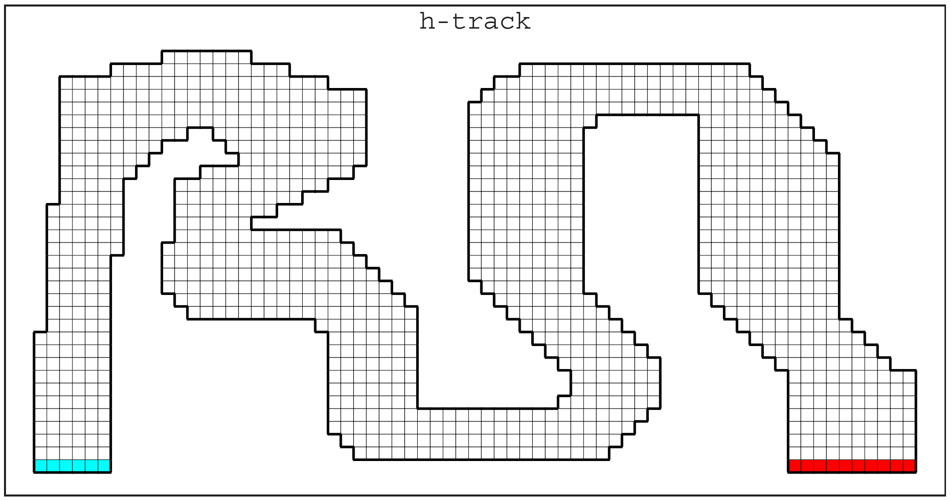
```
<action>        :=  NAME <action-body> |
                    NAME( <var-list> ) <typed-list> <action-body>

<action-body>   :=  <action-prec> | <action-effect> |
                    <action-cost> | <action-obs>

<action-prec>   :=  prec: <formula> | ϵ

<action-effect> :=  <det-eff> | <ndet-eff> | <stoc-eff> | ϵ
<det-eff>       :=  effect: <effect-list>
<ndet-eff>      :=  <det-eff> | <det-eff> <ndet-eff>
<stoc-eff>      :=  FLOAT <det-eff> |
                    FLOAT <det-eff> <stoc-eff>

<effect-list>   :=  <effect> | <effect> , <effect-list>
<effect>        :=  REL(<term-list>) | ¬REL(<term-list>) |
                    <term> := <term> |
                    when <formula> do <effect-list> end-do |
                    foreach <typed-var> do <effect-list> end-do

<action-cost>   :=  cost: <cost> | ϵ
<cost>          :=  INTEGER | <cost> + <cost> |
                    if <formula> <cost> <cost> fi |
                    (sum <typed-var> <cost>)

<action-obs>    :=  <det-obs> | <stoc-obs> | ϵ
<det-obs>       :=  obs: <obs-list>
<stoc-obs>      :=  FLOAT <det-obs> |
                    FLOAT <det-obs> <stoc-obs>
<obs-list>      :=  <obs> | <obs> , <obs-list>
<obs>           :=  <formula> | <term>


<formula>       :=  REL(<term-list>) |
                    (<term> PREL <term>) |
                    ¬<formula> |
                    (<formula> ∧ <formula>) |
                    (<formula> ∨ <formula>) |
                    (exists <typed-var> <formula>) |
                    (forall <typed-var> <formula>)

<term-list>     :=  <term> | <term> , <term-list>
<term>          :=  CONST | FUN(<term-list>) | <term> POP <term>

<var-list>      :=  VAR | VAR , <var-list>
<typed-var>     :=  (VAR - TYPE)
<typed-list>    :=  <var-list> - TYPE |
                    <var-list> - TYPE , <typed-list>


<ramification>  :=  NAME <ram-body> |
                    NAME( <var-list> ) <typed-list> <ram-body>
<ram-body>      :=  ramif: <effect-list>

<invariant>     :=  NAME <inv-body> |
                    NAME( <var-list> ) <typed-list> <inv-body>
<inv-body>      :=  inv: <formula>

<defined>       :=  NAME <def-body> |
                    NAME( <var-list> ) <typed-list> <def-body>
<def-body>      :=  defined: <formula>


<goal>          :=  goal: <goal-formula>
<goal-formula>  :=  <formula> | full-knowledge

<init>          :=  init: <init-eff-list>
<init-eff-list> :=  <init-eff> | <init-eff> , <init-eff-list>
<init-eff>      :=  <effect> | <term> := { <term-list> } |
                    prune <formula>
```

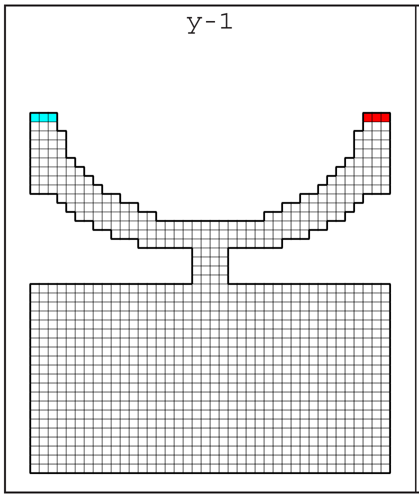
After parsing a defined predicate, the set of relational symbols is exteneded accordingly. The language is strongly typed. The terminals symbols PREL and POP refer to predefined relational and functional symbols like ‘$+$’, ‘$-$’, ‘$<$’, etc.

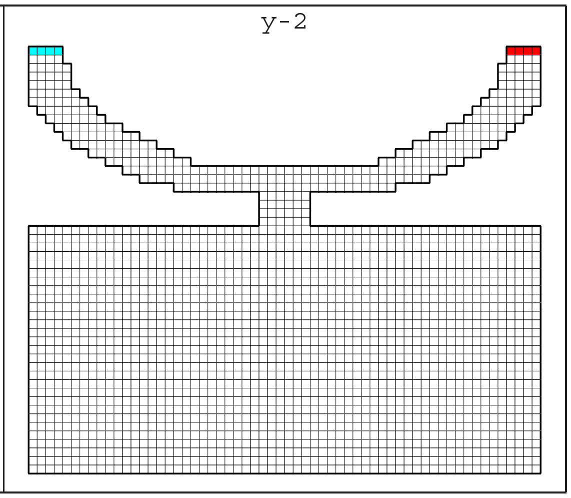
Appendix C

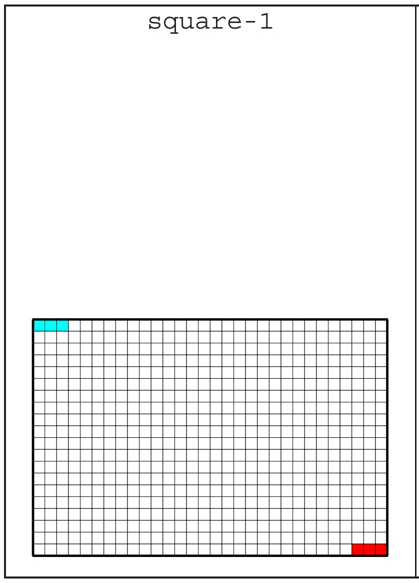
Racetrack Layouts

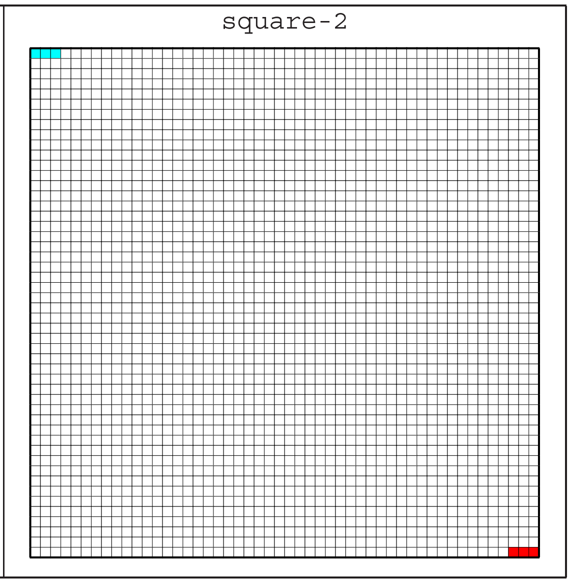


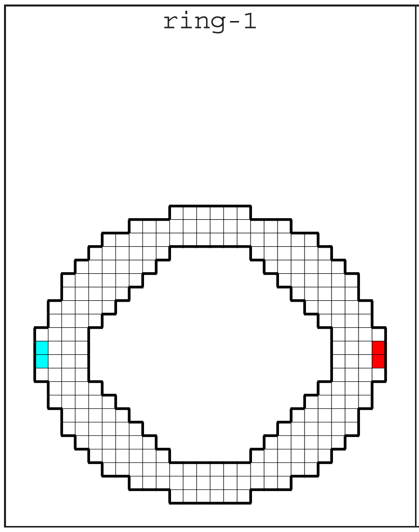


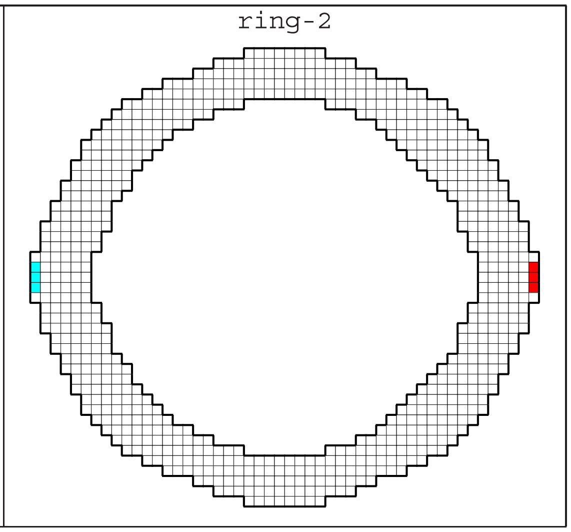

y-1

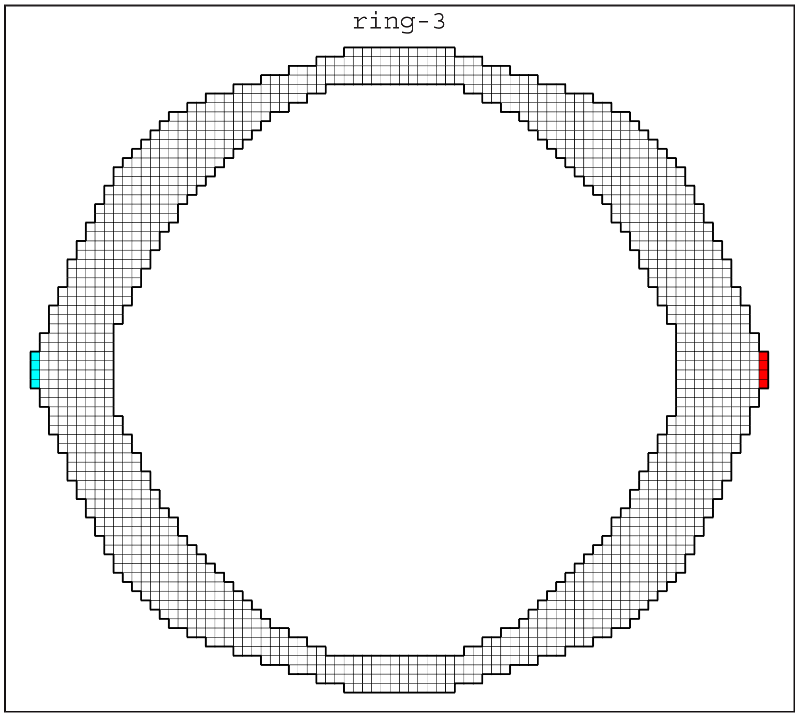

y-2

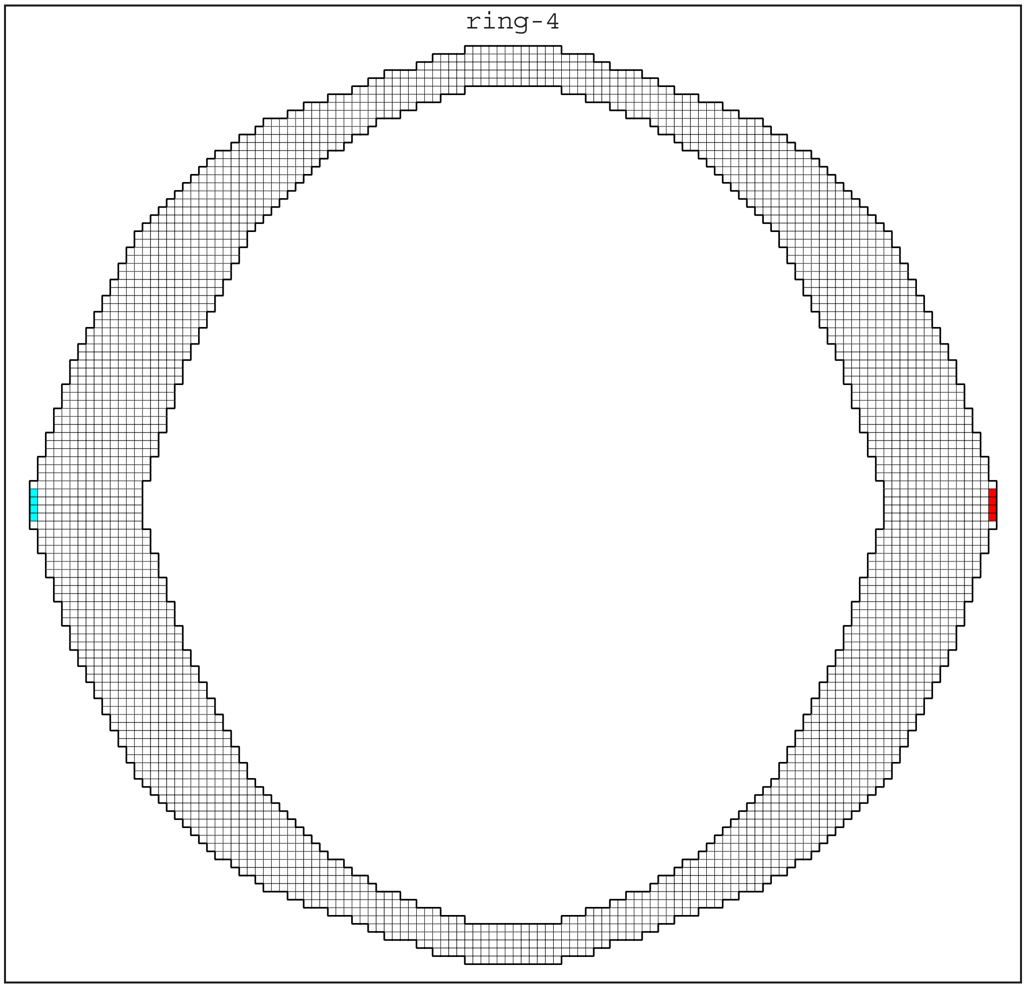

square-1


square-2


ring-1


ring-2


ring-3


ring-4

References

[1] And now, Master Mind. *Time*, 106(22):73, December 1, 1975.

[2] E. Adams. *The Logic of Conditionals*, chapter 2. D. Reidel, Dordrecht, Netherlands, 1975.

[3] P. Agre and D. Chapman. What are plans for? *Robotics and Autonomous Systems*, 6:17–34, 1990.

[4] K. Apt, H. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann, Los Altos, CA, 1987.

[5] K. Astrom. Optimal control of Markov Decision Processes with incomplete state estimation. *J. Math. Anal. Appl.*, 10:174–205, 1965.

[6] F. Bacchus. The 2000 AI Planning Systems Competition. *Artificial Intelligence Magazine*, 22, 2001.

[7] F. Bacchus and F. Kabanza. Using temporal logic to express search control knowledge for planning. *Artificial Intelligence*, 116(1–2):123–191, 2000.

[8] P. Bachmann. *Die analytische Zahlentheorie*. Teubner, Leipzig, 1894.

[9] A. Barto, S. Bradtke, and S. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72:81–138, 1995.

[10] V. Bayer-Zubek and T. Dietterich. Two heuristics for solving POMDPs having delayed need to observe. In A. Cimatti, H. Geffner, E. Giunchiglia, and J. Rintanen, editors, *Proc. IJCAI-01 Workshop on Planning with Uncertainty and Partial Information*, pages 51–60, Seattle, WA, 2001.

[11] V. Bayer-Zubek and T. Dietterich. Pruning improves heuristic search for cost-sensitive learning. In C. Sammut and A. Hoffmann, editors, *Proc. 19th International Conf. on Machine Learning*, pages 27–34, Sydney, Australia, 2002. Morgan Kaufmann.

[12] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[13] P. Bertoli, A. Cimatti, and M. Roveri. Heuristic search + symbolic model checking = efficient conformant planning. In B. Nebel, editor, *Proc. 17th International Joint Conf. on Artificial Intelligence*, pages 467–472, Seattle, WA, 2001. Morgan Kaufmann.

[14] P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Planning in nondeterministic domains under partial observability via symbolic model checking. In B. Nebel, editor, *Proc. 17th International Joint Conf. on Artificial Intelligence*, pages 473–478, Seattle, WA, 2001. Morgan Kaufmann.

[15] P. Bertoli, A. Cimatti, J. Slaney, and S. Thiébaux. Solving power supply restoration problems with planning via symbolic model-checking. In F. Van Harmelen, editor, *Proc. 15th European Conf. on Artificial Intelligence*, pages 576–580, Lyon, France, 2002. IOS Press.

[16] D. Bertsekas. *Dynamic Programming and Optimal Control, (2 Vols)*. Athena Scientific, 1995.

[17] D. Bertsekas and J. Tsitsiklis. An analysis of stochastic shortest-path problems. *Mathematics of Operations Research*, 16:580–595, 1991.

[18] D. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[19] P. Billingsley. *Probability and Measure*. Wiley-Interscience, third edition, 1995.

[20] C. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[21] A. Blum and M. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997.

[22] B. Bonet. A calculus for causal relevance. In J. Breese and D. Koller, editors, *Proc. 17th Conf. on Uncertainty in Artificial Intelligence*, pages 40–47, Seattle, WA, 2001. Morgan Kaufmann.

[23] B. Bonet. An ϵ-optimal grid-based algorithm for Partially Observable Markov Decision Processes. In C. Sammut and A. Hoffmann, editors, *Proc. 19th International Conf. on Machine Learning*, pages 51–58, Sydney, Australia, 2002. Morgan Kaufmann.

[24] B. Bonet and H. Geffner. Learning sorting and decision trees with POMDPs. In J. Shavlik, editor, *Proc. 15th International Conf. on Machine Learning*, pages 73–81, Madison, WI, 1998. Morgan Kaufmann.

[25] B. Bonet and H. Geffner. Planning as heuristis search: New results. In S. Biundo and M. Fox, editors, *Proc. 5th European Conf. on Planning*, pages 359–371, Durham, UK, 1999. Springer: Lecture Notes on Computer Science.

[26] B. Bonet and H. Geffner. Planning with incomplete information as heuristic search in belief space. In S. Chien, S. Kambhampati, and C. Knoblock, editors, *Proc. 6th International Conf. on Artificial Intelligence Planning and Scheduling*, pages 52–61, Breckenridge, CO, 2000. AAAI Press.

[27] B. Bonet and H. Geffner. GPT: a tool for planning with uncertainty and partial information. In A. Cimatti, H. Geffner, E. Giunchiglia, and J. Rintanen, editors, *Proc. IJCAI-01 Workshop on Planning with Uncertainty and Partial Information*, pages 82–87, Seattle, WA, 2001.

[28] B. Bonet and H. Geffner. Heuristic search planner 2.0. *Artificial Intelligence Magazine*, 22(3):77–80, Fall 2001.

[29] B. Bonet and H. Geffner. Planning and control in artificial intelligence: A unifying perspective. *Applied Intelligence*, 14(3):237–252, 2001.

[30] B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1–2):5–33, 2001.

[31] B. Bonet and H. Geffner. Faster heuristic search algorithms for planning with uncertainty and full feedback. In G. Gottlob, editor, *Proc. 18th International Joint Conf. on Artificial Intelligence*, pages 1233–1238, Acapulco, Mexico, 2003. Morgan Kaufmann.

[32] B. Bonet and H. Geffner. Labeled RTDP: Improving the convergence of real-time dynamic programming. In E. Giunchiglia, N. Muscettola, and D. Nau, editors, *Proc. 13th International Conf. on Automated Planning and Scheduling*, pages 12–21, Trento, Italy, 2003. AAAI Press.

[33] B. Bonet, G. Loerincs, and H. Geffner. A robust and fast action selection mechanism for planning. In B. Kuipers and B. Webber, editors, *Proc. 14th National Conf. on Artificial Intelligence*, pages 714–719, Providence, RI, 1997. AAAI Press / MIT Press.

[34] B. Bonet and J. Pearl. Qualitative MDPs and POMDPs: An order-of-magnitude approximation. In A. Darwiche and N. Friedman, editors, *Proc. 18th Conf. on Uncertainty in Artificial Intelligence*, pages 61–68, Edmonton, Canada, 2002. Morgan Kaufmann.

[35] B. Bonet and S. Thiébaux. GPT meets PSR. In E. Giunchiglia, N. Muscettola, and D. Nau, editors, *Proc. 13th International Conf. on Automated Planning and Scheduling*, pages 102–111, Trento, Italy, 2003. AAAI Press.

[36] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

[37] C. Boutilier, R. Dearden, and M. Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1–2):49–107, 2000.

[38] C. Boutilier and D. Poole. Computing optimal policies for partially observable decision processes using compact representations. In W. Clancey and D. Weld, editors, *Proc. 13th National Conf. on Artificial Intelligence*, pages 1168–1175, Portland, OR, 1996. AAAI Press / MIT Press.

[39] C. Boutilier, R. Reiter, and B. Price. Symbolic dynamic programming for first-order MDPs. In B. Nebel, editor, *Proc. 17th International Joint Conf. on Artificial Intelligence*, pages 690–697, Seattle, WA, 2001. Morgan Kaufmann.

[40] R. Brafman. A heuristic variable grid solution for POMDP's. In B. Kuipers and B. Webber, editors, *Proc. 14th National Conf. on Artificial Intelligence*, pages 727–733, Providence, RI, 1997. AAAI Press / MIT Press.

[41] R. Brooks. A robust layered control system for a mobile robot. *IEEE J. of Robotics and Automation*, 2:14–27, 1987.

[42] R. E. Bryant. Symbolic boolean manipulation with ordered bynaru-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[43] A. Cassandra, M. Littman, and L. Kaelbling. Acting optimally in partially observable stochastic domains. In B. Hayes-Roth and R. Korf, editors, *Proc. 12th National Conf. on Artificial Intelligence*, pages 1023–1028, Seattle, WA, 1994. AAAI Press / MIT Press.

[44] A. Cassandra, M. Littman, and N. Zhang. Incremental pruning: A simple, fast, exact algorithm for Partially Observable Markov Decision Processes. In D. Geiger, P. Shenoy, and P. Pundali, editors, *Proc. 13th Conf. on Uncertainty in Artificial Intelligence*, pages 54–61, Providence, RI, 1997. Morgan Kaufmann.

[45] C. C. Chang and H. J. Keisler. *Model Theory*. Elsevier Science, Amsterdam, 1990.

[46] P. Chatalic and L. Simon. Multi-resolution on compressed sets of clauses. In *Proc. 12th IEEE International Conf. on Tools with Artificial Intelligence*, pages 449–454, Vancouver, Canada, 2000.

[47] H. T. Cheng. *Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, University of British Columbia, 1988.

[48] C. Chow and J. Tsitsiklis. An optimal one-way multigrid algorithm for continuous-state discrete-time stochastic control. *IEEE Trans. on Automatic Control*, 36(8):898–914, 1991.

[49] L. Chrisman. Reinforcement learning with perceptual aliasing: the perceptual distinctions approach. In P. Rosenbloom and P. Szolovits, editors, *Proc. 10th National Conf. on Artificial Intelligence*, pages 183–188, San Jose, CA, 1992. AAAI Press / MIT Press.

[50] V. Chvatal. Mastermind. *Combinatorica*, 3(3–4):325–329, 1983.

[51] A. Cimatti and M. Roveri. Conformant planning via symbolic model checking. *Journal of Artificial Intelligence Research*, 13:305–338, 2000.

[52] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[53] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[54] J. Culberson and J. Schaeffer. Pattern databases. *Computational Intelligence*, 14(3):318–334, 1998.

[55] A. Darwiche and M. Goldszmidt. On the relation between Kappa calculus and probabilistic reasoning. In R. Lopez de Mantaras and D. Poole, editors, *Proc. 10th Conf. on Uncertainty in Artificial Intelligence*, pages 145–153, Seattle, WA, 1994. Morgan Kaufmann.

[56] A. Darwiche and P. Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

[57] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, 1962.

[58] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[59] T. Dean, L. Kaelbling, J. Kirman, and A. Nicholson. Planning with deadlines in stochastic domains. In R. Fikes and W. Lehnert, editors, *Proc. 11th National Conf. on Artificial Intelligence*, pages 574–579, Washington, D.C., 1993. AAAI Press / MIT Press.

[60] T. Dean, L. Kaelbling, J. Kirman, and A. Nicholson. Planning under time constraints in stochastic domains. *Artificial Intelligence*, 76(1–2):35–74, 1995.

[61] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1-2):41–85, 1999.

[62] R. Dechter and J. Pearl. Generalized best-first search strategies and the optimality of A*. *Journal of the ACM*, 32(3):505–536, 1985.

[63] R. Dechter and I. Rish. Directional Resolution: The Davis Putnam procedure, revisited. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Proc. 4th International Conf. on Principles of Knowledge Representation and Reasoning*, pages 134–145, Bonn, Germany, 1994. Morgan Kaufmann.

[64] A. del Val. A new method for consequence finding and compilation in restricted languages. In J. Hendler and D. Subramanian, editors, *Proc. 16th National Conf. on Artificial Intelligence*, pages 259–264, Orlando, FL, 1999. AAAI Press / MIT Press.

[65] E. Denardo. Contraction mappings in the theory underlying dynamic programming. *SIAM Review*, 9:165–177, 1967.

[66] E. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[67] M. Do and S. Kambhampati. Solving planning-graph by compiling it into a csp. In S. Chien, S. Kambhampati, and C. Knoblock, editors, *Proc. 6th International Conf. on Artificial Intelligence Planning and Scheduling*, pages 82–91, Breckenridge, CO, 2000. AAAI Press.

[68] P. Doherty, J. Gustafsson, L. Karlsson, and J. Kvarnstrom. (TAL) temporal action logics: Language specification and tutorial. *Electronic Transactions on Artificial Intelligence*, 2(3–4):273–306, 1998.

[69] D. Draper, S. Hanks, and D. Weld. Probabilistic planning with information gathering and contingent execution. In K. Hammont, editor, *Proc. 2nd International Conf. on Artificial Intelligence Planning Systems*, pages 31–36, Chicago, IL, 1994. AAAI Press.

[70] D. Dubois and H. Prade. *Possibility Theory: An Approach to Computerized Processing and Uncertainty*. Plenum Press, New York, NY, 1988.

[71] D. Dubois and H. Prade. Possibility theory as a basis for qualitative decision theory. In C. Mellish, editor, *Proc. 14th International Joint Conf. on Artificial Intelligence*, pages 1925–1930, Montreal, Canada, 1995. Morgan Kaufmann.

[72] R. Durrett. *Probability: Theory and Examples*. Duxbury Press, second edition, 1995.

[73] J. Eagle. The optimal search for a moving target when search path is constrained. *Operations Research*, 32(5):1107–1115, 1984.

[74] H. Enderton. *A Mathematical Introduction to Logic*. Academic Press, second edition, 2000.

[75] P. Erdös and C. Ranyi. On two problems in information theory. *Magyar Tud. Akad. Mat. Kut. Int. Közl.*, 8:229–242, 1963.

[76] O. Etzioni. Acquiring search-control knowledge via static analysis. *Artificial Intelligence*, 62(2):255–302, 1993.

[77] O. Etzioni, S. Hanks, D. Draper, N. Lesh, and M. Williamson. An approach to planning with incomplete information. In B. Nebel, C. Rich, and W. Swartout, editors, *Proc. 3rd International Conf. on Principles of Knowledge Representation and Reasoning*, pages 115–125, Boston, MA, 1992. Morgan Kaufmann.

[78] S. Even. *Graph Algorithms*. Computer Science Press, 1979.

[79] R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning about Knowledge*. MIT Press, 1995.

[80] H. Fargier, J. Lang, and R. Sabbadin. Towards qualitative approaches to multi-stage decision making. *Int. Journal of Approximate Reasoning*, 19:441–471, 1998.

[81] Z. Feng and E. Hansen. Symbolic heuristic search for factored markov decision processes. In R. Dechter, M. Kearns, and R. Sutton, editors, *Proc. 18th National Conf. on Artificial Intelligence*, pages 455–460, Edmonton, Canada, 2002. AAAI Press / MIT Press.

[82] R. Fikes and N. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 1:27–120, 1971.

[83] M. Flood. Mastermind strategy. *J. Recreational Mathematics*, 18(3):194–202, 1985–86.

[84] R. Fourer, D. Gay, and B. Kernighan. *AMPL: A modeling language for mathematical programming*. The Scientific Press, 1993.

[85] M. Fox and D. Long. The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research*, 9:367–421, 1998.

[86] M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. See `http://www.dur.ac.uk/d.p.long/competition.html`, 2001.

[87] J. B. Fraleigh. *A First Course in Abstract Algebra*. Addison-Wesley, Reading, MA, sixth edition, 1999.

[88] B. Fristedt and L. Gray. *A Modern Approach to Probability Theory*. Birkhauser, 1997.

[89] H. Geffner. *Default Reasoning: Causal and Conditional Theories*. MIT Press, 1992.

[90] H. Geffner. Functional STRIPS. In J. Minker, editor, *Logic-Based Artificial Intelligence*, pages 187–205. Kluwer, 2000.

[91] H. Geffner and B. Bonet. Solving large pomdps using real time dynamic programming, 1998. Working Notes Fall AAAI Symposium on POMDPS.

[92] H. Geffner and J. Wainer. Modeling action, knowledge and control. In H. Prade, editor, *Proc. 13th European Conf. on Artificial Intelligence*, pages 532–536, Brighton, UK, 1998. Wiley.

[93] M. Gelfond and V. Lifschitz. Representing action and change by logic programs. *J. of Logic Programming*, 17:301–322, 1993.

[94] M. Genesereth and I. Nourbakhsh. Time-Saving tips for problem solving with incomplete information. In R. Fikes and W. Lehnert, editors, *Proc. 11th National Conf. on Artificial Intelligence*, pages 724–730, Washington, D.C., 1993. AAAI Press / MIT Press.

[95] P. Giang and P. Shenoy. A qualitative linear utility theory for Spohn's theory of epistemic beliefs. In C. Boutilier and M. Goldszmidt, editors, *Proc. 16th Conf. on Uncertainty in Artificial Intelligence*, pages 220–229, Stanford, CA, 2000. Morgan Kaufmann.

[96] M. Goldszmidt and J. Pearl. System Z^+: A formalization for reasoning with variable strength defaults. In T. Dean and K. McKeown, editors, *Proc. 9th National Conf. on Artificial Intelligence*, pages 399–404, Anaheim, CA, 1991. AAAI Press / MIT Press.

[97] M. Goldszmidt and J. Pearl. Qualitative probabilities for default reasoning, belief revision, and causal modeling. *Artificial Intelligence*, 84:57–112, 1996.

[98] J. Halpern. Axiomatizing causal reasoning. *Journal of Artificial Intelligence Research*, 12:317–337, 2000.

[99] E. Hansen and S. Zilberstein. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129:35–62, 2001.

[100] 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.

[101] W. Harvey and M. Ginsberg. Limited discrepancy search. In C. Mellish, editor, *Proc. 14th International Joint Conf. on Artificial Intelligence*, pages 607–613, Montreal, Canada, 1995. Morgan Kaufmann.

[102] P. Haslum and H. Geffner. Admissible heuristic for optimal planning. In S. Chien, S. Kambhampati, and C. Knoblock, editors, *Proc. 6th International Conf. on Artificial Intelligence Planning and Scheduling*, pages 140–149, Breckenridge, CO, 2000. AAAI Press.

[103] P. Haslum and H. Geffner. Heuristic planning with time and resources. In A. Cesta, editor, *Proc. 6th European Conf. on Planning*, Toledo, Spain, 2001. Springer: Lecture Notes on Computer Science.

[104] M. Hauskrecht. Value-function approximations for partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, 13:33–94, 2000.

[105] J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier. SPUDD: Stochastic planning using decision diagrams. In K. Laskey and H. Prade, editors, *Proc. 15th Conf. on Uncertainty in Artificial Intelligence*, pages 279–288, Stockholm, Sweden, 1999. Morgan Kaufmann.

[106] J. Hoffmann. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research*. accepted for the special issue on the 3rd International Planning Competition.

[107] J. Hoffmann and H. Geffner. Branching matters: Alternative branching in Graphplan. In E. Giunchiglia, N. Muscettola, and D. Nau, editors, *Proc. 13th International Conf. on Automated Planning and Scheduling*, pages 22–31, Trento, Italy, 2003. AAAI Press.

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

[109] N. Hyafil and F. Bacchus. Conformant probabilistic planning via CSPs. In E. Giunchiglia, N. Muscettola, and D. Nau, editors, *Proc. 13th International Conf. on Automated Planning and Scheduling*, pages 205–214, Trento, Italy, 2003. AAAI Press.

[110] R. W. Irving. Towards an optimun mastermind strategy. *J. Recreational Mathematics*, 11(2):81–87, 1978–79.

[111] F. V. Jensen. *An Introduction to Bayesian Networks*. Springer, New York, 1996.

[112] A. Junghanns and J. Schaeffer. Sokoban: Enhancing general single-agent search methods using domain knowledge. *Artificial Intelligence*, 129:219–251, 2001.

[113] L. Kaelbling, M. Littman, and A. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1999.

[114] L. Kaelbling, M. Littman, and A. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[115] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In W. Clancey and D. Weld, editors, *Proc. 13th National Conf. on Artificial Intelligence*, pages 1194–1201, Portland, OR, 1996. AAAI Press / MIT Press.

[116] H. Kautz and B. Selman. Unifying SAT-based and Graph-based planning. In T. Dean, editor, *Proc. 16th International Joint Conf. on Artificial Intelligence*, Stockholm, Sweden, 1999.

[117] D. E. Knuth. *The Art of Computer Programming, Vol. III: Sorting and Searching*. Addison-Wesley, 1973.

[118] D. E. Knuth. The computer as a Master Mind. *J. Recreational Mathematics*, 9:1–6, 1976-1977.

[119] S. Koenig. Minimax real-time heuristic search. *Artificial Intelligence*, 129:165–197, 2001.

[120] R. Korf. Depth-first iterative-depeening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.

[121] R. Korf. Real-time heuristic search. *Artificial Intelligence*, 42(2–3):189–211, 1990.

[122] R. Korf. Linear-space best-first search. *Artificial Intelligence*, 62(1):41–78, 1993.

[123] R. Korf. Finding optimal solutions to rubik's cube using pattern databases. In B. Kuipers and B. Webber, editors, *Proc. 14th National Conf. on Artificial Intelligence*, pages 700–705, Providence, RI, 1997. AAAI Press / MIT Press.

[124] R. Korf and A. Felner. Disjoint pattern database heuristics. *Artificial Intelligence*, 134:9–22, 2002.

[125] R. Korf, M. Reid, and S. Edelkamp. Time complexity of iterative-deepening-a*. *Artificial Intelligence*, 129(1–2):199–218, 2001.

[126] R. Korf and L. Taylor. Finding optimal solutions to the twenty-four puzzle. In W. Clancey and D. Weld, editors, *Proc. 13th National Conf. on Artificial Intelligence*, pages 1202–1207, Portland, OR, 1996. AAAI Press / MIT Press.

[127] S. Kraus, D. Lehmann, and M. Magidor. Non-monotonic reasoning, preferential models and cummulative logics. *Artificial Intelligence*, 44(1–2):167–207, 1990.

[128] S. Kripke. Semantical considerations on modal logic. In L. Linsky, editor, *Reference and Modality*, pages 63–72. Oxford University Press, 1971.

[129] N. Kushmerick, S. Hanks, and D. Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, 76:239–286, 1995.

[130] E. Landau. *Handbuch der Lehre von der Verteilung der Primzahlen*. Teubner, Leipzig, 1909.

[131] A. Leitsch. *The Resolution Calculus*. Springer, Berlin, Germany, 1997.

[132] H. Levesque. What is planning in the presence of sensing. In W. Clancey and D. Weld, editors, *Proc. 13th National Conf. on Artificial Intelligence*, pages 1139–1146, Portland, OR, 1996. AAAI Press / MIT Press.

[133] M. Littman. Probabilistic propositional planning: representations and complexity. In B. Kuipers and B. Webber, editors, *Proc. 14th National Conf. on Artificial Intelligence*, pages 748–754, Providence, RI, 1997. AAAI Press / MIT Press.

[134] M. Littman, A. Cassandra, and L. Kaelbling. Learning policies for partially observable environments: Scaling up. In A. Prieditis and S. Russell, editors, *Proc. 12th International Conf. on Machine Learning*, pages 362–370, Tahoe, CA, 1995. Morgan Kaufmann.

[135] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1984.

[136] W. Lovejoy. Computationally feasible bounds for partially observed Markov decision processes. *Operations Research*, 39, 1991.

[137] W. Lovejoy. A survey of algorithmic techniques for partially observed Markov decision processes. *Annals of Operations Research*, 28:47–66, 1991.

[138] C. Lusena, M. Mundhenk, and J. Goldsmith. Nonapproximability results for partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, 14:83–103, 2001.

[139] S. Majercik and M. Littman. Maxplan: A new approach to probabilistic planning. In R. Simmons, M. Veloso, and S. Smith, editors, *Proc. 4th International Conf. on Artificial Intelligence Planning Systems*, pages 86–93, Pittsburgh, PA, 1998. AAAI Press.

[140] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In T. Dean and K. McKeown, editors, *Proc. 9th National Conf. on Artificial Intelligence*, pages 634–639, Anaheim, CA, 1991. AAAI Press / MIT Press.

[141] A. McCallum. Overcoming incomplete perception with utile distinction memory. In P. Utgoff, editor, *Proc. 10th International Conf. on Machine Learning*, pages 190–196, Amherst, MA, 1993. Morgan Kaufmann.

[142] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer, D. Michie, and M. Swann, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, Edinburgh, Scotland, 1969.

[143] D. McDermott. A heuristic estimator for means ends analysis in planning. In B. Drabble, editor, *Proc. 3rd International Conf. on Artificial Intelligence Planning Systems*, pages 150–157, Edinburgh, Scotland, 1996. AAAI Press.

[144] D. McDermott. Official results for the first planning competition held at AIPS–98. See `http://ftp.cs.-yale.edu/pub/mcdermott/aipscomp-results.html`, 1998.

[145] D. McDermott. PDDL – The Planning Domain Definition Language, version 1.2. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, New Haven, CT, 1998.

[146] N. Meuleau and D. Smith. Optimal limited contingency planning. In C. Meek and U. Kjaerulff, editors, *Proc. 19th Conf. on Uncertainty in Artificial Intelligence*, Acapulco, Mexico, 2003. Morgan Kaufmann.

[147] S. Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proc. 30th Conf. ACM/IEEE Design Automation*, pages 272–277, 1993.

[148] S. Minton. *Learning Search Control Knowledge*. Kluwer Academic, Dordrecht, 1988.

[149] T. Miura and T. Ishida. Stochastic node caching for memory-bounded search. In C. Rich and J. Mostow, editors, *Proc. 15th National Conf. on Artificial Intelligence*, pages 450–456, Madison, WI, 1998. AAAI Press / MIT Press.

[150] G. Monahan. A survey of partially observable Markov decision processes: Theory, models and algorithms. *Management Science*, 28(1):1–16, 1983.

[151] A. Newell and H. Simon. GPS, a program that simulates human thought. In H. Billing, editor, *Lernende Automaten*, pages 109–124. R. Oldenbourg, Munich, Germany, 1961.

[152] A. Newell and H. Simon. *Human Problem Solving*. Prentice–Hall, Englewood Cliffs, NJ, 1972.

[153] N. Nilsson. *Principles of Artificial Intelligence*. Tioga, 1980.

[154] L. Padulo and M. Arbib. *System Theory*. Hemisphere Publishing Co., 1974.

[155] J. Pearl. *Heuristics*. Morgan Kaufmann, 1983.

[156] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

[157] J. Pearl. Epsilon semantics. In *Encyclopedia of AI*, pages 468–475. John Wiley & Sons, Inc., 2nd edition, 1992.

[158] J. Pearl. From conditional oughts to qualitative decision theory. In D. Heckerman and A. Mamdani, editors, *Proc. 9th Conf. on Uncertainty in Artificial Intelligence*, pages 12–22, Washington, D.C., 1993. Morgan Kaufmann.

[159] J. Pearl. From Adams' conditionals to default expressions, causal conditionals, and counterfactual. In E. Eells and B. Skyrms, editors, *Probability and Conditionals*. Cambridge University Press, 1994.

[160] J. Pearl. *Causality: Models, Rreasoning and Inferece*. Cambridge University Press, New York, 2000.

[161] E. Pednault. ADL: Exploring the middle ground between Strips and the situation calculus. In R. Brachman, H. Levesque, and R. Reiter, editors, *Proc. 1st International Conf. on Principles of Knowledge Representation and Reasoning*, pages 324–332, Toronto, Canada, 1989. Morgan Kaufmann.

[162] M. Peot and D. Smith. Conditional nonlinear planning. In J. Hendler, editor, *Proc. 1st International Conf. on Artificial Intelligence Planning Systems*, pages 189–197, College Park, MD, 1992. Morgan Kaufmann.

[163] L. Pryor and G. Collins. Planning for contingencies: A decision-based approach. *Journal of Artificial Intelligence Research*, 4:287–339, 1996.

[164] M. Puterman. *Markov Decision Processes – Discrete Stochastic Dynamic Programming*. John Wiley and Sons, Inc., 1994.

[165] J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kauffman, 1993.

[166] R. Reiter. On closed-world data bases. In H. Gallaire and J. Minker, editors, *Logic and data bases*, pages 55–76. Plenum Press, 1978.

[167] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 12:81–132, 1980.

[168] R. Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press, 1991.

[169] R. Reiter. *Knowledge in Action: Logical foundations for describing and implementing dynamical systems*. MIT Press, 2001.

[170] J. Rintanen. Constructing conditional plans by a theorem prover. *Journal of Artificial Intelligence Research*, 10:323–352, 1999.

[171] J. Rintanen. Expressive equivalence of formalisms for planning with sensing. In E. Giunchiglia, N. Muscettola, and D. Nau, editors, *Proc. 13th International Conf. on Automated Planning and Scheduling*, pages 185–194, Trento, Italy, 2003. AAAI Press.

[172] A. Robinson. *Non-Standard Analysis*. Princeton Landmarks in Mathematics, reprinted edition, 1996.

[173] J. A. Robinson. A machine oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

[174] H. L. Royden. *Real Analysis*. Collier MacMillan, second edition, 1968.

[175] W. Rudin. *Real and Complex Analysis*. McGraw-Hill Science, third edition, 1986.

[176] S. Russell. Efficient memory-bounded search methods. In B. Neumann, editor, *Proc. 10th European Conf. on Artificial Intelligence*, pages 1–5, Vienna, Austria, 1992. Wiley.

[177] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1994.

[178] R. Sabbadin. A possibilistic model for qualitative sequential decision problems under uncertainty in partially observable environments. In K. Laskey and H. Prade, editors, *Proc. 15th Conf. on Uncertainty in Artificial Intelligence*, pages 567–574, Stockholm, Sweden, 1999. Morgan Kaufmann.

[179] E. Sandewal. *Features and Fluents. The Representation of Knowledge about Dynamical Systems*. Oxford University Press, 1994.

[180] R. Schachter. Evaluating influence diagrams. *Operations Research*, 34:871–882, 1986.

[181] B. Selman and H. Kautz. Knowledge compilation and theory approximation. *Journal of the ACM*, 43:193–224, 1996.

[182] Y. Shoham and D. McDermott. Problems in formal temporal logics reasoning. *Artificial Intelligence*, 36(1):49–61, 1988.

[183] L. Simon and A. del Val. Efficient consequence finding. In B. Nebel, editor, *Proc. 17th International Joint Conf. on Artificial Intelligence*, pages 359–365, Seattle, WA, 2001. Morgan Kaufmann.

[184] D. J. Slate and L. R. Atkin. CHESS 4.5 – The Northwestern University chess program. In P. W. Frey, editor, *Chess Skill in Man and Machine*, pages 82–118. Springer-Verlag, Berlin, Germany, 1977.

[185] D. Smith and D. Weld. Conformant planning. In C. Rich and J. Mostow, editors, *Proc. 15th National Conf. on Artificial Intelligence*, pages 889–896, Madison, WI, 1998. AAAI Press / MIT Press.

[186] E. Sondik. *The Optimal Control of Partially Observable Markov Decision Processes*. PhD thesis, Stanford University, 1971.

[187] E. Sondik. The optimal control of partially observable Markov decision processes over the infinite horizon: discounted costs. *Operations Research*, 26(2), 1978.

[188] P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction, and Search*. MIT Press, second edition, 2001.

[189] W. Spohn. A general non-probabilistic theory of inductive reasoning. In R. Schachter, T. Levitt, L. Kanal, and J. Lemmer, editors, *Proc. 4th Conf. on Uncertainty in Artificial Intelligence*, pages 149–158, New York, NY, 1988. North-Holland.

[190] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[191] R. E. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[192] J. Tash and S. Russell. Control strategies for a stochastic planner. In B. Hayes-Roth and R. Korf, editors, *Proc. 12th National Conf. on Artificial Intelligence*, pages 1079–1085, Seattle, WA, 1994. AAAI Press / MIT Press.

[193] L. Taylor and R. Korf. Pruning duplicate nodes in depth-first search. In R. Fikes and W. Lehnert, editors, *Proc. 11th National Conf. on Artificial Intelligence*, pages 756–761, Washington, D.C., 1993. AAAI Press / MIT Press.

[194] S. Thiébaux and M. Cordier. Supply restoration in power distribution systems – a benchmark for planning under uncertainty. In A. Cesta, editor, *Proc. 6th European Conf. on Planning*, pages 85–95, Toledo, Spain, 2001. Springer: Lecture Notes on Computer Science.

[195] S. Thiébaux, M. Cordier, O. Jehl, and J. Krivine. Supply restoration in power distribution systems – a case study in integrating model-based diagnosis and repair planning. In E. Horvitz and F. Jensen, editors, *Proc. 12th Conf. on Uncertainty in Artificial Intelligence*, pages 525–532, Portland, OR, 1996. Morgan Kaufmann.

[196] S. Thiébaux, J. Hoffmann, and B. Nebel. In defense of PDDL axioms. In G. Gottlob, editor, *Proc. 18th International Joint Conf. on Artificial Intelligence*, pages 961–966, Acapulco, Mexico, 2003. Morgan Kaufmann.

[197] P. Tison. Generalized consensus theory and application to the minimization of boolean circuits. *IEEE Trans. on Computers*, EC-16:446–456, 1967.

[198] P. Van Hentenryck. *The OPL optimization programming language*. MIT Press, 1999.

[199] T. Vossen, M. Ball, A. Lotem, and D. Nau. On the use of integer programming models in AI planning. In T. Dean, editor, *Proc. 16th International Joint Conf. on Artificial Intelligence*, Stockholm, Sweden, 1999.

[200] D. Weld. An introduction to least commitment planning. *Artificial Intelligence Magazine*, 1994.

[201] N. Wilson. An order of magnitude calculus. In P. Besnard and S. Hanks, editors, *Proc. 11th Conf. on Uncertainty in Artificial Intelligence*, pages 548–555, Montreal, Canada, 1995. Morgan Kaufmann.

[202] R. Yager, S. Ovchinnikov, S. Tong, and H. Nguyen, editors. *Fuzzy Sets and Applications: Selected papers by L. A. Zadeh*. Wiley, New York, NY, 1987.

[203] N. Zhang and S. Lee. Planning with Partially Observable Markov Decision Processes: Advances in exact solution methods. In G. Cooper and S. Moral, editors, *Proc. 14th Conf. on Uncertainty in Artificial Intelligence*, pages 523–530, Madison, WI, 1998. Morgan Kaufmann.

[204] N. Zhang and W. Liu. A model approximation scheme for planning in partially observable stochastic domains. *Journal of Artificial Intelligence Research*, 7:199–230, 1997.

[205] N. Zhang and W. Zhang. Speeding up the convergence of Value Iteration in Partially Observable Markov Decision Processes. *Journal of Artificial Intelligence Research*, 14:29–51, 2001.