Policy Explanation and Model Refinement in Decision-Theoretic Planning

by

Omar Zia Khan

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2013

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Decision-theoretic systems, such as Markov Decision Processes (MDPs), are used for sequential decision-making under uncertainty. MDPs provide a generic framework that can be applied in various domains to compute optimal policies. This thesis presents techniques that offer explanations of optimal policies for MDPs and then refine decision theoretic models (Bayesian networks and MDPs) based on feedback from experts.

Explaining policies for sequential decision-making problems is difficult due to the presence of stochastic effects, multiple possibly competing objectives and long-range effects of actions. However, explanations are needed to assist experts in validating that the policy is correct and to help users in developing trust in the choices recommended by the policy. A set of domain-independent templates to justify a policy recommendation is presented along with a process to identify the minimum possible number of templates that need to be populated to completely justify the policy.

The rejection of an explanation by a domain expert indicates a deficiency in the model which led to the generation of the rejected policy. Techniques to refine the model parameters such that the optimal policy calculated using the refined parameters would conform with the expert feedback are presented in this thesis. The expert feedback is translated into constraints on the model parameters that are used during refinement. These constraints are non-convex for both Bayesian networks and MDPs. For Bayesian networks, the refinement approach is based on Gibbs sampling and stochastic hill climbing, and it learns a model that obeys expert constraints. For MDPs, the parameter space is partitioned such that alternating linear optimization can be applied to learn model parameters that lead to a policy in accordance with expert feedback.

In practice, the state space of MDPs can often be very large, which can be an issue for real-world problems. Factored MDPs are often used to deal with this issue. In Factored MDPs, state variables represent the state space and dynamic Bayesian networks model the transition functions. This helps to avoid the exponential growth in the state space associated with large and complex problems. The approaches for explanation and refinement presented in this thesis are also extended for the factored case to demonstrate their use in real-world applications. The domains of course advising to undergraduate students, assisted hand-washing for people with dementia and diagnostics for manufacturing are used to present empirical evaluations.

Acknowledgments

Pascal Poupart, was my mentor and supervisor for the last six years. I am forever indebted to him for taking me under his guidance and teaching me how to approach research with singular focus and persistence, always making himself available when I needed his assistance and feedback, and for his constant understanding, patience and support when I started working before the completion of this work. I could not have asked for a better supervisor.

This work originally began with Jay Black, who remained my co-supervisor for a large portion of this work while he was at Waterloo. I am grateful to him for always allowing me the freedom to pursue areas of my interest and insisting on clarity of thought in technical exposition. Jay has also been a constant source of sound technical and personal advice. I am fortunate to have spent time with him.

I thank John Mark Agosta for allowing me to frame my research in terms of a real-world problem by providing me with access to an invaluable data set. I have benefited immensely by observing his attention to detail and commitment to stay abreast of new research.

I am grateful to my thesis committee: Robin Cohen and Dan Brown for their detailed and critical review of this work, Derek Koehler for helping me draw connections of this work with research in psychology, and Judy Goldsmith for being my external examiner and carefully reading my thesis. Their recommendations have only made this thesis better.

My colleagues from the AI Lab and Shoshin Lab also deserve a mention here. The weekly seminars in the AI Lab allowed me to broaden my research horizons while the members of Shoshin provided me with a desk, even after Jay had left.

Margaret Towell in the Computer Science Grad Office has always been an invaluable resource who was untiring in her readiness to assist me whenever I needed any help in navigating the maze of the paperwork necessary for all Ph.D. students at Waterloo.

I would also like to acknowledge that funding for portions of this research came from Natural Sciences and Engineering Research Council of Canada (NSERC), David R. Cheriton School of Computer Science, University of Waterloo, and Intel Corporation.

I formed several life-lasting friendships during my stay at Waterloo that only made this period of life more memorable. The one that I'll cherish the most is the one with Kamran. Thank you for being my listening board on all things life, making sure I remained grounded, and teaching me how to give without ever expecting anything back in return.

My parents have offered me their unconditional love, support and prayers for everything in life that I have ever pursued. None of this would have been possible without them. They will remain my role models for parenthood and I can only hope to do for them a fraction of what they did for me. Thank you.

Dedication

To my life partner, Farheen, for being as accommodating and understanding as she is to all my shortcomings and quirkiness.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In many situations, a sequence of decisions must be taken by an individual or system (*e.g.*, course selection by students, diagnosing a fault in a machine, assisting a person with dementia in everyday tasks). Traditionally, these tasks are handled by human experts (student advisors provide advice to students regarding course selection, technicians diagnose faults, and caregivers prompt elderly with dementia to help them complete their tasks). Domain experts can be assisted by providing them with an intelligent automated assistant, in the form a recommender system, that can suggest the best available choice for a situation. When experts are not available, these recommender systems can also be used to make recommendations directly to end-users or execute actions on their behalf. Decision-theoretic planning provides a framework through which the optimal choice in a given situation can be calculated. Such an approach allows harnessing the computational power of machines to optimize difficult scenarios but it also results in the issue of trust. Recommendations that have been automatically generated by machines may not be readily accepted by users. If an explanation can also be generated, in addition to calculating the optimal choice, it may help users in accepting the recommendation. In some domains (such as diagnostics), there is also a need to incorporate feedback from experts to adjust the model since sufficient data (used to develop the models) can never be collected, and repeatedly making sub-optimal recommendations is not acceptable. This thesis addresses these issues of automated policy explanation and model refinement in decision-theoretic planning, more specifically policy explanation for Markov decision processes (MDPs) and model refinement for Bayesian networks and MDPs.

1.1 Motivation

Decision theory [80] provides a framework to evaluate different actions in a given state. The actions are evaluated using a transition function and a reward function. The transition function provides a distribution of the possible outcomes over each action in a given state whereas the reward function encodes the preferences for each possible outcome. The objective of decision-theoretic planning [15] then is to provide a process through which the action with the maximum possible reward can be chosen. This suggests that transition and reward functions need to be specified or learnt to compute optimal policies.

Reinforcement learning [89] is used when the transition and reward functions are not known a priori. Reinforcement learning learns the dynamics of the system (transition and reward functions) by balancing the exploration-exploitation trade-off. The exploration step is concerned with acquiring new samples to learn more about the effects of different actions and the preferences of the user. This can lead to the execution of sub-optimal actions. The exploitation step utilizes knowledge from previous exploration to select the action that is believed to provide the maximum expected reward. In most domains, it is not acceptable to allow the system to explore and execute sub-optimal actions due to an incomplete model. If the system is providing advice to a human, exploration may not be possible at all. If humans feel the recommendation is sub-optimal, they may never execute that action and the system will never be able to learn that it is sub-optimal. This can happen even if an action recommended by the system is actually optimal for a state, as the human may not necessarily agree with this choice. Also, if the observations are dependent on a human executing an action, often it may not be possible to collect enough data to learn the model in any case. Finally, if a user is presented a recommendation and she ignores it in favor of an alternate action, her assumption may be that the system will learn not to make the mistake again which is not obvious under the reinforcement learning paradigm. The system repeating the same mistakes will lead to user frustration.

Uncertainty in the effect of actions and complex objectives make understanding the optimality of an action notoriously difficult. This can lead to two key issues. First, designers find it difficult to create accurate models or to otherwise debug them. It is difficult to design MDPs because real-world MDPs often have hundreds of thousands, if not millions of states. There is a need for tools for experts to examine and/or debug their models. The current design process involves successive iterations of tweaking various parameters to achieve a desirable output. At the end, the experts still cannot verify if the policy is indeed reflecting their requirements accurately.

Second, users find it difficult to understand the reasoning behind an automated recommendation. Users also have to trust the policy by treating it as a black box, with no

explanations whatsoever regarding the process of computing the recommendation or the confidence of the system in this recommendation. They cannot observe which factors have been considered by the system while making the recommendation. The system also cannot accept suggestions from the user to modify the policy and update the model to reflect this preference.

Both of these issues are among key bottlenecks to the widespread use of decision-theoretic planners such as Markov decision processes (MDPs) [79]. Thus, there is a need for explanations that enhance understanding of model to establish user trust in the MDP recommendations. Similarly, there is also a need for automated refinement of models when users disagree with the optimal choice of the model and recommend an alternate choice. Explanations in this environment will help validate the models and any technique to correct the model such that it reflects their desired outcome would speed up the process of creating accurate MDP models.

In this thesis, techniques to explain MDP policies and refine models for Bayesian networks [72] and MDPs based on feedback from experts are presented. Just as MDPs can be solved using techniques like value iteration without any changes specific to domains, the intent here is to develop techniques to explain MDP policies in different domains without any domain-specific changes as well refine models generating these policies based on feedback from experts in a generic manner.

1.2 Contributions

The major contributions of this thesis are as follows:

- *Explanations for MDP Policies.* Previous approaches to generating explanations for experts in intelligent systems involve presenting users with reasoning traces. However, similar techniques cannot be used for MDPs. The techniques required to solve MDPs involve mathematical optimization so the complete numerical process cannot be presented as an execution trace. The challenge here is to abstract the process of solving the policy, yet retain sufficient information for experts, so that they can comprehend the computation to detect any inconsistencies or errors and be reasonably confident of the accuracy of their models. Chapter 3 presents an approach to present explanations for MDP policies in a domain-independent fashion. These explanations are presented by filling in templates that provide information about the consequences of executing an action. A method to select the minimum number of templates needed to justify an action is also described.

- *Refinement of Parameters for Bayesian Networks.* Chapter 4 presents an approach to refine models for Bayesian networks by incorporating non-convex constraints obtained from expert feedback. Previously, various techniques have been presented that incorporate constraints from domain knowledge but these techniques assume linear constraints. The constraint arising from an expert choosing one action over another is non-convex and cannot be incorporated using prior work. The approach presented in this chapter augments Bayesian networks by explicitly modeling the constraint in the network. Gibbs sampling is then used on this augmented network to compute a maximum a posteriori (MAP) estimate and obtain a refined model.

- *Refinement of Parameters for Markov Decision Processes.* The process of learning user preferences and consequently the reward function in an MDP has been studied under preference elicitation and inverse reinforcement learning paradigms. The problem of learning the reward function can be posed as a linear optimization problem if the transition function is known. On the other hand, if the reward function is known and the transition function needs to be learned, the problem is non-convex. Chapter 5 presents an approach, based on alternating linear optimization, that allows learning the transition function with expert feedback.

Explanations for MDP optimal policies will provide insight into the reasoning of the system and explain why the optimal action is the best action given the transition function specified for the MDP. This will help MDP designers understand if the model has been specified correctly and provide end-users some level of justification that may increase their trust in the recommendation from the MDP. The refinement of Bayesian networks and MDPs will provide a process for designers of these models to refine them such that the recommendations from the refined models match the choices passively observed from domain experts.

Domain experts are not necessarily assumed to be well-versed with probabilistic approaches such as Bayesian networks or MDPs. However, they are considered experts for the domain the recommender system is being built for. This means that if an expert is passively observed, the recommendations of the expert can be treated as the correct behavior that the recommender system should mimic. The refinement techniques presented in Chapters 4 and 5 are based on taking advantage of feedback from domain experts.

1.3 Thesis Outline

The thesis is structured as follows. Chapter 2 describes the relationship between decision theory, probability theory and utility theory. Various models such as Bayesian networks, influence diagrams and Markov decision processes are formally introduced. In practice, real-world problems often require the use of factored models so the concept of factored MDPs is also discussed. Example domains of course advising and diagnostics are also explained in this chapter, as they will be used as running examples through the thesis.

Chapter 3 focuses on the problem of explaining MDP policies. A review of existing literature on explanations for intelligent systems is presented to analyze the type of explanations that have been historically used to explain automated reasoning and planning. More specifically, the literature on explanations for Bayesian networks, influence diagrams and MDPs is also reviewed. Then, a set of templates for MDP policy explanations are presented that can be populated at run-time in a domain independent fashion. A technique to select the minimum number of templates required to sufficiently explain an optimal action is also presented for both the flat and factored cases. Sample explanations are presented for the hand-washing and course advising domains. The results of a user study conducted to evaluate the course advising explanations are also discussed in this chapter.

Chapter 4 presents a solution to the problem of model refinement for Bayesian networks based on expert feedback. First, the constraints obtained from experts are formally defined. Then, the existing literature on refining parameters for Bayesian networks is reviewed and it is shown that expert constraints cannot be incorporated using prior approaches due to non-convexity of these constraints. A solution, based on Gibbs sampling, is presented to tackle the problem of refining the parameters by using these constraints. Two variants of the technique are discussed, one with rejection sampling and the other without any rejection. The techniques are then evaluated on synthetic and real-world diagnostic Bayesian networks obtained from a manufacturing domain.

Chapter 5 focuses on the issue of refining the transition functions of MDPs based on expert feedback. A review of existing literature on MDP model refinement (transition and reward functions) is presented. The problem of model refinement is then formalized by representing the feedback from experts as constraints on the model. A technique based on alternating linear optimization is presented to refine the parameters for flat MDPs. The concepts of policy graphs and Monte Carlo value iteration, that have been previously proposed for partially observable MDPs, are adapted for use with factored MDPs. This is necessary to demonstrate the scalability of factored MDPs on large scale problems. A technique is then described that enables model refinement of large scale factored MDPs

by using policy graphs and Monte Carlo value iteration. Experimental results are then evaluated on synthetic and real-world diagnostic MDPs obtained from a manufacturing scenario.

Finally, Chapter 6 concludes by presenting a summary of the contributions of this thesis and discusses some directions for future work.

Chapter 2

Background

This chapter introduces the concept of decision-theoretic planning by referring to probability and utility theory. Different methods for probabilistic reasoning are then discussed. Techniques such as Bayesian networks and dynamic Bayesian networks are introduced as they are used to represent the dynamics of the model. Techniques such as influence diagrams and Markov decision processes are introduced as they are used for selecting optimal actions. This chapter also introduces the domains of course advising and diagnosis that are used as sample domains to demonstrate the techniques of policy explanation and model refinement presented in the rest of the thesis.

2.1 Decision Theory and Decision-Theoretic Planning

Decision theory [80] deals with identifying optimal actions under uncertain conditions. It draws upon principles from probability theory and utility theory [97] to compute optimal policies. Probability theory allows representation of uncertainty while utility theory allows specification of preferences.

Probabilities are used for the dual purposes of dealing with uncertainty about the current state, and dealing with uncertainty in the effect of a particular action. If the state is represented by a random variable X, then $Pr(X = x)$ denotes the probability of the current state being x, or in other words, the degree of belief that the current state is x. A belief state represents a probability distribution over all possible states or all possible values of X. The prior probability distribution on X can be represented as $Pr(X)$. The prior represents the belief state, before any evidence is received. Evidence provides information

about observed values of X or other random variables. If evidence e is observed, then the new belief can be represented using $Pr\left(X|e\right)$ and is known as the posterior probability distribution. Beliefs are updated with the accumulation of more evidence using Bayes' Theorem, shown in Equation 2.1.

$$Pr\left(X|e\right) = \frac{Pr\left(e|X\right) Pr\left(X\right)}{Pr\left(e\right)} \tag{2.1}$$

where $Pr\left(e|X\right)$ represents the probability of observing evidence e given the belief about X and $Pr\left(e\right)$ is a normalization constant that can be computed as $Pr\left(e\right) = \sum_X Pr\left(e|X\right) Pr(X)$. Thus, prior probabilities are used to compute posterior probabilities when new evidence is available. Bayes' Theorem is used widely in decision-theoretic systems for probabilistic inference.

Probabilities only encode uncertainty about the likelihood of occurrence of events, and do not provide any information regarding the desirability of those events. Utilities (also known as rewards or costs) are used to express preferences across different events. Utility can be composed of multiple attributes, with each attribute expressing preference over different aspects of an event or an outcome. In such cases, it is generally assumed that utility is additive, i.e., utilities for all attributes are added.

The expected utility of an outcome is computed by using Equation 2.2.

$$EU = \sum_{i=1}^{n} Pr\left(s_i\right) U\left(s_i\right) \tag{2.2}$$

where $s_i \in S$ and S is the set of all possible outcomes, $|S| = n$, $Pr\left(s_i\right)$ is the probability of s_i occurring, and $U\left(s_i\right)$ is the utility of outcome s_i. Thus, the expected utility is computed by combining information regarding the desirability and the likelihood of occurrence of each event or outcome. The principle of maximum expected utility underlines the basic assumption of utility theory. The principle states that a rational agent will always act to maximize its expected utility. Decision-theoretic systems are also based on this principle and choose the action that maximizes expected utility. It is also known that the optimal action computed using expected utility remains invariant under positive linear transformations [97]. Thus, the designers can choose any scale to express their preferences as desired, without impacting the optimal decision.

MDPs can be well understood using the context of probabilistic systems such as Bayesian networks and dynamic Bayesian networks, and decision-theoretic systems such as influ-

ence diagrams, so these techniques are briefly introduced next before formally introducing MDPs.

2.2 Bayesian Networks

A Bayesian network [72] is a directed acyclic graph in which each node represents a random variable and each arc represents a conditional dependency. The node from which the arc originates is called the parent of the node on which the arc terminates. A conditional probability distribution is associated with each node that quantifies the effect of all its parents on it. For discrete variables, the conditional probability distribution can be represented as a conditional probability table (CPT). Figure 2.1 shows a graphical representation of a Bayesian network. All nodes in the network are assumed to have binary values.

Bayesian networks provide information about direct dependencies between different variables, and hence can be used to perform inference. They can be used to calculate the full joint distribution using Equation 2.3, where $Parents\left(X_i\right)$ denotes the actual values of the parents of the variable X_i.

$$Pr\left(X_1 = x_1, \ldots, X_n = x_n\right) = \prod_{i=1}^{n} Pr\left(x_i | Parents\left(X_i\right)\right) \tag{2.3}$$

Thus, a Bayesian network can be considered as a compact representation of the joint probability distribution which can then be used to answer different queries using some available evidence. Techniques such as variable elimination [103], that use Bayes' Theorem, are used for inference in Bayesian networks.

2.2.1 Diagnostic Bayesian Networks

Diagnostic Bayesian networks [7] are used to model the incorrect behavior of a system that can be caused by one or more possible causes. Diagnostic Bayesian networks are used in Chapter 4 to demonstrate the technique to refine models for Bayesian networks.

A class of bipartite Bayesian networks is widely used as diagnostic models where the network forms a sparse, directed, causal graph, where arcs go from causes to observable node variables. Upper case is used denote random variables; C for causes, and T for observables (tests). Lower case letters denote values in the domain of a variable, e.g. $c \in dom(C) = \{c, \bar{c}\}$, and bold letters denote sets of variables. A set of marginally

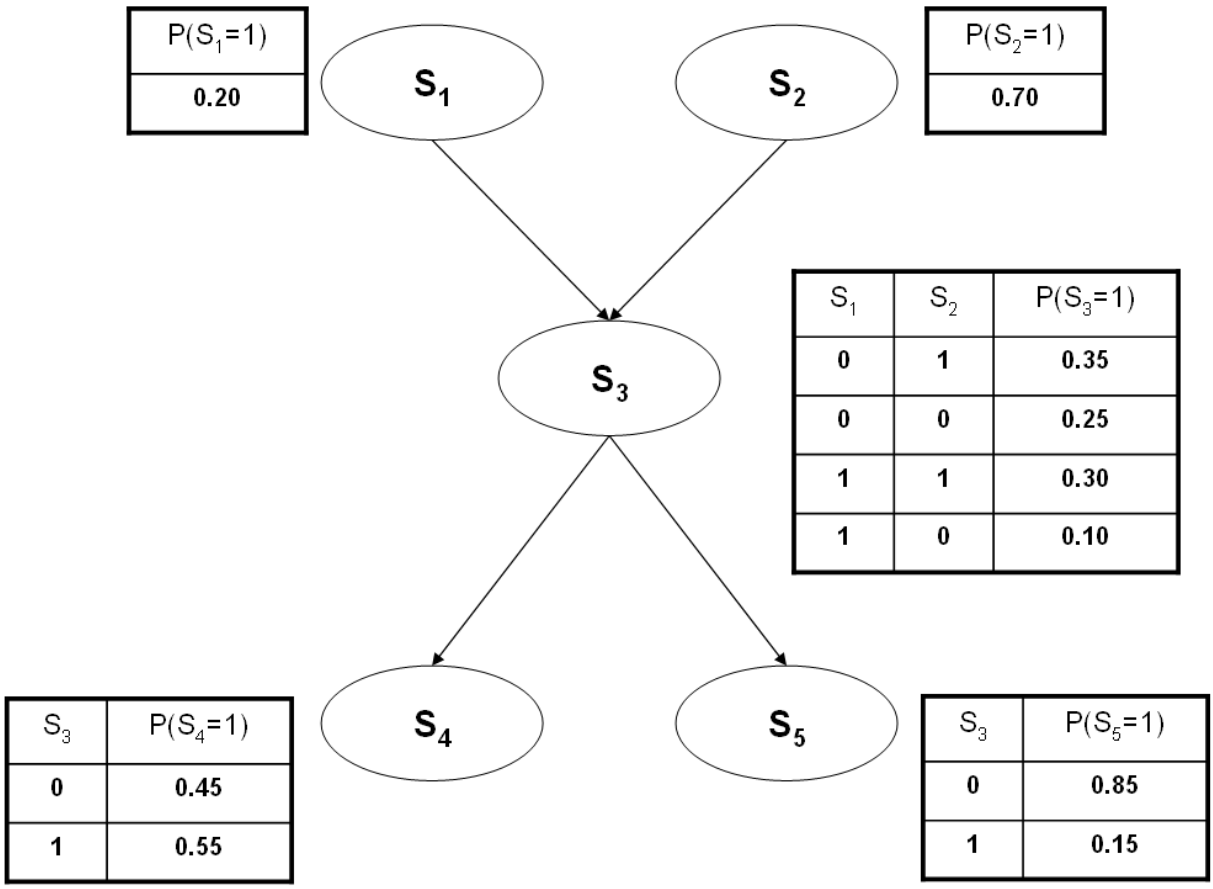


Figure 2.1: A graphical representation of a Bayesian network.

independent binary-valued test variables $\mathbf{C}$ with distributions $\Pr(C)$ represent unobserved causes, and condition the remaining conditionally independent binary-valued test variable nodes $\mathbf{T}$. Each cause conditions one or more tests; likewise each test is conditioned by one or more causes, resulting in a graph with one or more possibly multiply-connected components. The test variable distributions $\Pr(T|C)$ incorporate the further modeling assumption of Independence of Causal Influence, the most common example being the Noisy-Or model [43]. To keep the exposition simple, it is assumed in this thesis that all variables are binary and that conditional distributions are parametrized by the Noisy-Or; however, the techniques described in the thesis generalize to any discrete non-binary variable models.

A *diagnostic sequence* is defined as consisting of the assignment of values to a subset of tests. The *diagnostic process* embodies the choice of the best next test to execute at each step in the sequence, by measuring the *diagnostic value* among the set of available tests at each step, that is, the ability of a test to distinguish among the possible causes.

Conventionally, unobserved tests are ranked in diagnosis by their Value Of Information (VOI) [48] conditioned on tests already observed. To be precise, VOI is the expected gain in utility if the test were to be observed. The complete computation requires a model equivalent to a partially observable Markov decision process. An alternate measure to rank tests is based on Gini impurity, also known as Gini index. This involves ordering the test variables based on the expected remaining uncertainty in the underlying cause, which can be measured by the Gini impurity as shown in Equation 2.4. The Gini impurity measures the expected error rate when $\mathbf{c}$ is chosen at random according to $\Pr(\mathbf{C} = \mathbf{c}|T = t)$.

$$\mathrm{GI}(\mathbf{C}|T) = \sum_t \Pr(T = t)\Big[\sum_{\mathbf{c}} \Pr(\mathbf{C} = \mathbf{c}|T = t)(1 - \Pr(\mathbf{C} = \mathbf{c}|T = t))\Big] \tag{2.4}$$

VOI can also be approximated by a greedy computation of the Mutual Information between a test and the set of causes [10].It has also been shown that Mutual Information can be well approximated to second order by the *Gini impurity* [38] and thus can be used as a surrogate for VOI, as a way to rank the best next test in the diagnostic sequence. In Chapter 4, Gini impurity will be used as a way to rank tests.

2.3 Dynamic Bayesian Networks

Dynamic Bayesian networks (DBNs) are an extension to Bayesian networks, that take into account evidence accumulated over time. In DBNs, just like Bayesian networks, certain

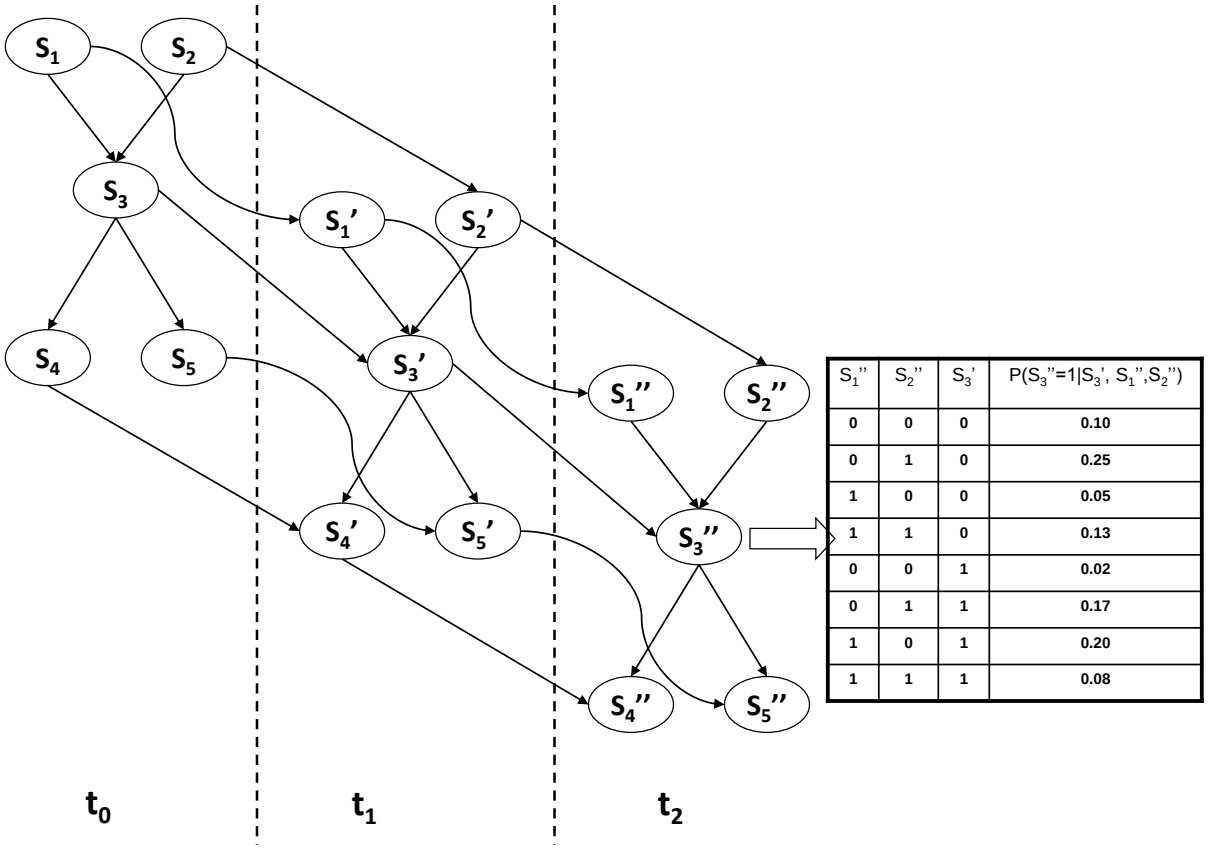


Figure 2.2: A graphical representation of a DBN.

variables are observable and the rest are hidden. However, the difference is that with the progression of time, different values for the observable variables are observed, which in turn influence the values of the hidden variables. DBNs employ the Markov property, according to which the probability distribution of hidden variables at time t is only dependent on their distribution at time $t-1$ and the observed evidence. Thus, there is a Bayesian network for each time slice. An additional assumption is that the model is stationary, *i.e.*, the conditional probabilities do not change over time. Figure 2.2 shows a graphical representation of a DBN. All nodes are assumed to have binary values.

The process of inferring a distribution over the current state given the distribution over the previous state and all evidence observed is known as filtering. Inference in DBNs

can be performed using the same techniques as in Bayesian networks, such as variable elimination, by unrolling the DBN into a Bayesian network. Unrolling creates a larger Bayesian network with each time slice replicated to incorporate all observed evidence.

DBNs provide a compact representation for the transition dynamics of a system. This representation will be used later to represent the transition function in an MDP.

2.4 Influence Diagrams

Influence diagrams [49], also known as decision networks, extend Bayesian networks to support decision-making. Figure 2.3 shows a graphical representation of an influence diagram. Ovals represent chance nodes, rectangles represent decision nodes, and diamonds represent utility nodes.

Influence diagrams contain two additional types of nodes called decision and utility nodes. Decision nodes represent possible actions whereas utility nodes represent the utility associated with a state. The random variables in Bayesian networks are represented in influence diagrams as chance nodes. The principle of maximum expected utility is used to choose the action that yields the highest expected value for the utility nodes. The expected utility is computed by performing inference and using variable elimination. Note that influence diagrams only provide a recommended action for a single step, and not a sequence of inter-related actions.

2.5 Markov Decision Processes:

Markov decision processes (MDP) [79] are decision-theoretic systems that are used for sequential decision making under uncertainty. While influence diagrams can model arbitrary configurations of decision points, MDPs assume a single sequence of decision points, i.e., the same distribution is assumed over each time step and the choice of executing an action has to be made repeatedly. MDPs are typically represented using DBNs and hence also employ the Markov property. Note that the expected utility in MDPs combines the expected utilities of all future actions.

An MDP is formally defined by a tuple $\{S, A, T, \rho, \gamma\}$, where

- S is a set of states s. A state is defined by the joint instantiation of a finite set of state variables.

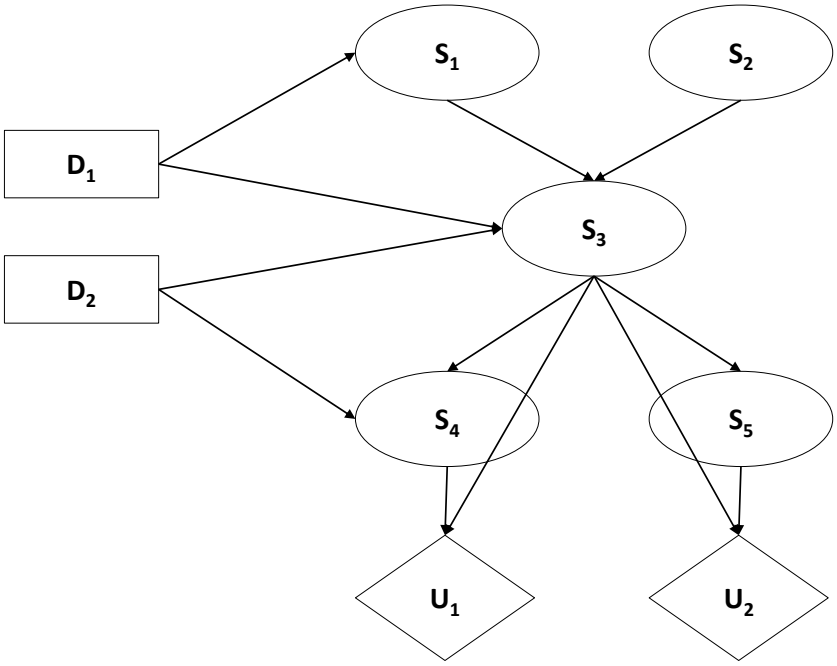


Figure 2.3: A graphical representation of an influence diagram.

- A is a set of actions a. It is assumed in this document that this set is finite.
- T represents the transition model, such that $T : S \times A \times S \rightarrow [0, 1]$ specifies the probability $Pr\left(s'|a, s\right)$ of an action a in state s leading to a state s'.
- ρ represents the reward model, such that $\rho : S \times A \rightarrow \mathbb{R}$ specifies the utility or reward $\rho\left(s, a\right)$ associated with executing action a in state s. The reward function can be defined by a set $\mathcal{R}$ of reward variables R such that the sum of their values r is the reward at any given state (e.g. $\rho\left(s, a\right) = \sum_{R \in \mathcal{R}} r_{R,s,a}$). Thus, it is assumed in this document that the reward function is additive and allows for separate encoding of different objectives.
- γ represents the discount factor, $0 \leq \gamma \leq 1$. A low value of the discount factor indicates that a reward accumulated earlier is preferred to that accumulated later. If the discount factor is 1, then it indicates that there is no preference for acquiring rewards sooner.

Figure 2.4 shows a graphical representation of an MDP.

An MDP policy $\pi : S \rightarrow A$ consists of a mapping from states to actions. The value $V^{\pi}\left(s\right)$ of a policy π when starting in state s is the sum of the expected discounted rewards earned while executing the policy. This is shown in Eq. 2.5, which is also known as the value function.

$$V^{\pi}\left(s_0\right) = E\left[\sum_{t=0}^{\infty} \gamma^t \rho\left(s_t, a_t\right) \middle| \pi, s_0\right] \tag{2.5}$$

The expectation here is over the transitions induced by following policy π. A policy can be evaluated by using Bellman's equation (Eq. 2.6) which measures the value of a state.

$$V^{\pi}\left(s\right) = \rho\left(s, \pi\left(s\right)\right) + \gamma \sum_{s'} T\left(s', a, s\right) V^{\pi}\left(s'\right) \tag{2.6}$$

An optimal policy π^* earns the highest value for all states (i.e., $V^{\pi^*}(s) \geq V^{\pi}(s)\ \forall, s, \pi \neq \pi^*$). Optimal policies for MDPs can be computed using techniques such as value iteration in which Bellman's optimality equation (Eq. 2.7) is treated as an update rule that is applied iteratively. Essentially, the utility of a state is determined by adding the immediate reward with the expected discounted utility of the next state determined by choosing the optimal action.

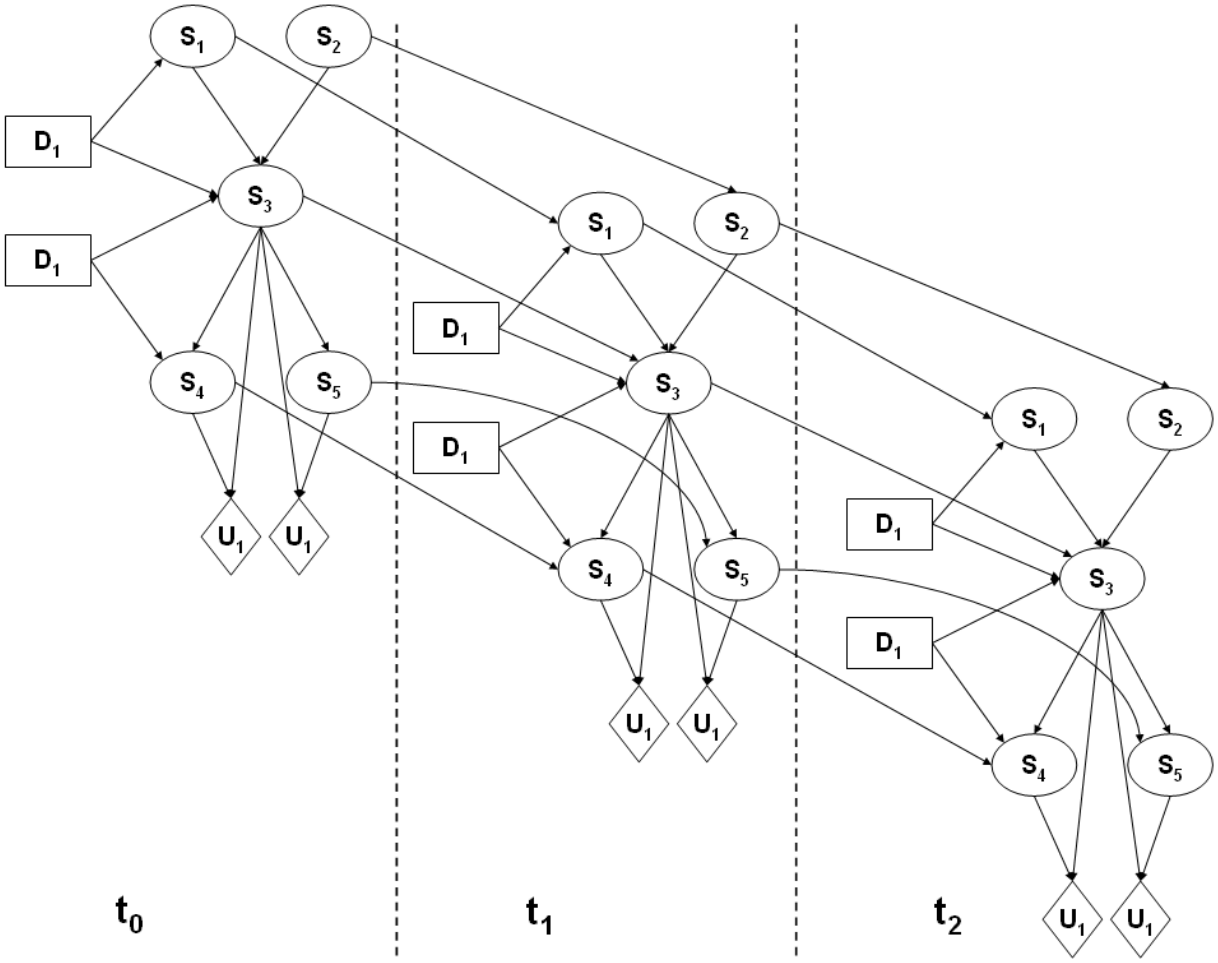


Figure 2.4: A graphical representation of an MDP

$$V^{\pi^*}(s) = \max_a \left[\rho(s,a) + \gamma \sum_{s'} T(s', a, s) V^{\pi^*}(s') \right] \tag{2.7}$$

Action-value functions [89], also known as Q-functions, represented as $Q^\pi(s,a)$, are used to evaluate the expected value of executing an action, a in a state s, and executing a policy, π, thereafter. This can be considered a function that assigns a value to every state-action pair and can be computed using Equation 2.8.

$$Q^\pi(s,a) = \rho(s,a) + \gamma \sum_{s'} T(s', a, s) V^\pi(s') \tag{2.8}$$

These concepts of value functions and Q-functions will be used through the rest of the thesis in policy explanation and model refinement for MDPs.

2.5.1 Factored MDPs

The state space for most real-world problems is often defined by the cross-product of the domain of several variables. Converting such problems to a traditional representation of MDPs defined in the previous section (also referred to as a flat representation) can result in exponential explosion in the state space and make the problem of solving for an optimal policy computationally prohibitive. In such a case, a factored representation is more suitable where the state space is instead represented by storing the value of each variable. Such MDPs are often referred to as factored MDPs [16] since the transition function is the product of several factors, each corresponding to a conditional distribution of a variable given its parents.

For this thesis, it is assumed that each state variable is discrete and can take a finite number of possible values. The current state of an MDP, s is then defined by the current values of all state variables. The transition function, T, for a factored MDP is a factored transition model in which the transitions can be represented by a DBN. This allows for a compact representation of the transition function and it is reasonable since actions typically affect only a subset of state variables. These effects also depend only on a subset of state variables and can thus be represented compactly using a DBN.

Since the state space is represented by instantiating each variable with a certain value, it is possible to define the concept of a scenario for factored MDPs. A scenario represents a collection of states that corresponds to the instantiation of a subset of the state variables.

The states in the scenario can then be obtained by taking a cross product of the values of the assigned and unassigned variables. The concept of a scenario will be used in Chapter 3 when explaining MDPs.

2.5.2 MDPs for Recommender Systems

In this thesis, the focus is on recommender systems where an MDP recommends to a user an action at each step. Examples of recommender MDPs include diagnostics, course advising, and so on. Recommender systems lend themselves naturally to a factored representation. The state contains one variable for the outcome of each action i.e., diagnostic tests or grades for courses. The actions are recommendations for the next diagnostic test to execute or the next course to register. It is assumed that repeating an action does not change the result, so no states are revisited in the MDP.

Figure 2.5 depicts the flat representation of a factored recommender MDP with three state variables $\{A_1, A_2, A_3\}$. Each node represents a state, each arc represents a transition from one state to another via an action corresponding to the label of the arc. In this example, the variables have domain $\{T, F, \_\}$. A null value, "_", for a variable indicates that this action has not yet been executed and thus its results has not yet been observed. The actions here are the recommendations to observe the value of a variable. More generally, recommender MDPs can be structured in a similar way with variables that can take n values corresponding to $n-1$ observations or the null value _.

It is clear from Figure 2.5 that transition functions for MDPs have a special structure that lends itself naturally to a factored representation. When an action (corresponding to observing a variable) is executed in a state, the new state will be the union of the previously observed values with the observed value of the action executed in this step.

The states in a recommender MDP can be organized in levels, where each level groups all the states with the same number of variables instantiated. For instance, in Figure 2.5 at level 0, no variable has been observed and only one state is part of this level. All actions are available at this level. At level 1, each state has one variable observed, so the number of actions available at Level 1 is two since the action corresponding to the observed variable is no longer available. Similarly, at level 3, three variables have been observed and no further actions are available with all variables already observed. This concept of levels can be used to enforce a partial ordering on the states such that all states in Level 0 are ordered lower than all states in Level 1, and all states in Level 1 are ordered lower than all states in Level 2, and so on. This ordering also makes it clear that states are never visited more than once in a recommender MDP. This is because an action once executed is never repeated. Thus,

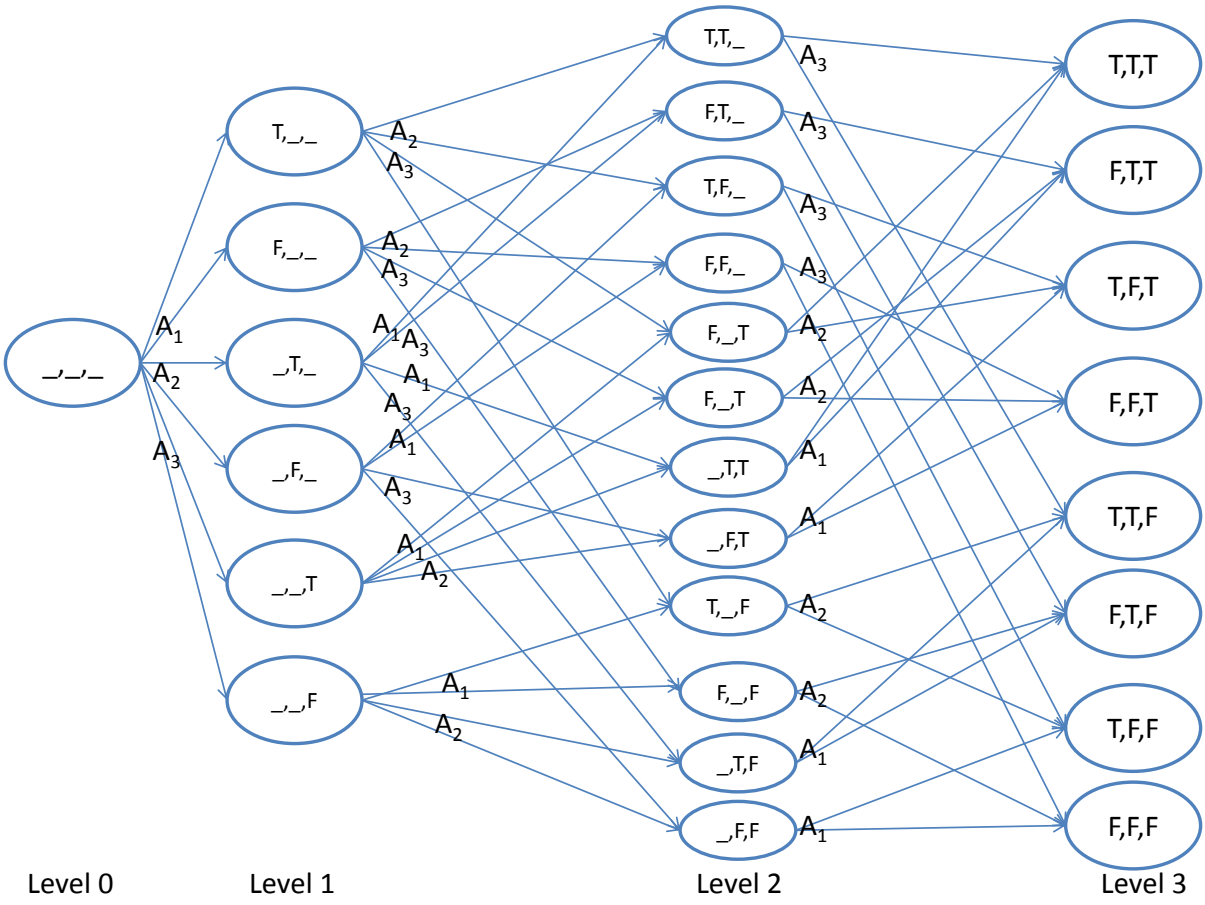


Figure 2.5: Sample flat recommender MDP

every time an action is executed the state transition results in a new previously unvisited state.

This thesis uses the domains of diagnostics and course-advising for the purpose of demonstrating the techniques presented for explanation and refinement. The next two sections explain how these domains can be modeled as recommender MDPs.

2.5.3 MDPs for Diagnostics

The policy for a diagnostic recommender MDP recommends the optimal action to execute at any given state, where an action may be executing a diagnostic test or predicting the underlying fault/cause. The reward function can be used to fulfill various criteria such as diagnosing the fault as quickly as possible, reducing the overall cost of executing costs, or minimizing the cost of mis-classification.

Bayer-Zubek and Dietterich [13] presented a model for diagnostic MDPs that is used in Chapter 5 to demonstrate the refinement of model parameters for an MDP. The total number of actions in such an MDP are the total number of tests that can be executed plus the action of predicting the cause. Once the cause has been predicted, the MDP terminates. In diagnostic MDPs, the cause is not directly observable. Since all variables in the MDP are observable, the MDP state does not explicitly represent the cause. But the reward function requires this information (high reward if correct diagnosis and vice versa). To account for this, the reward function is represented using a misclassification cost, $MC(f_k, y)$, that represents the cost of predicting cause k when the true cause is y. Since the correct diagnosis is not available in the state, the actual cost of executing an action can be viewed as a random variable that takes the value $MC\left(f_k, y\right)$, with probability $Pr(y|s)$. The expected cost of executing an action can then be computed as shown in Equation 2.9.

$$C\left(s, f_k\right) = \sum_y Pr\left(y|s\right) \cdot MC\left(f_k, y\right) \tag{2.9}$$

Consider the MDP represented in Figure 2.6 with two tests and two possible causes. The state space of this factored MDP has three variables, one each corresponding to the tests and the third for the prediction of the cause. Once the MDP has predicted a cause, it terminates.

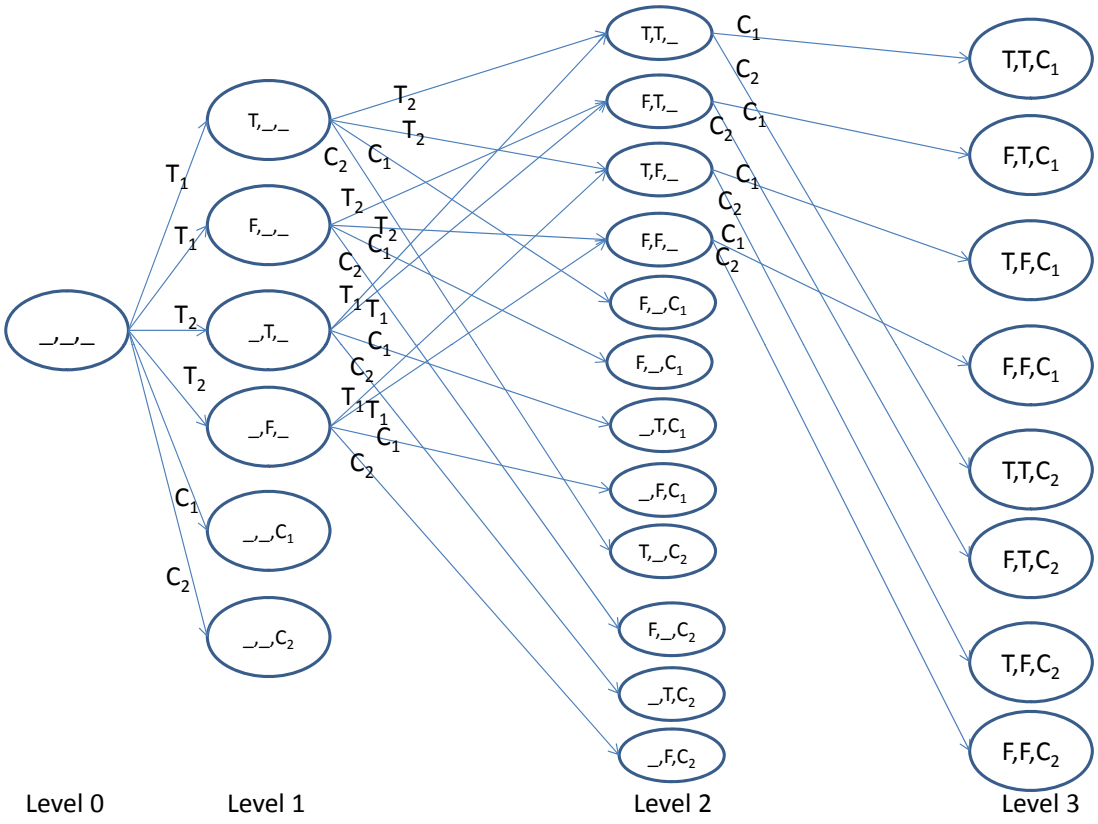


Figure 2.6: Sample diagnostic recommender MDP

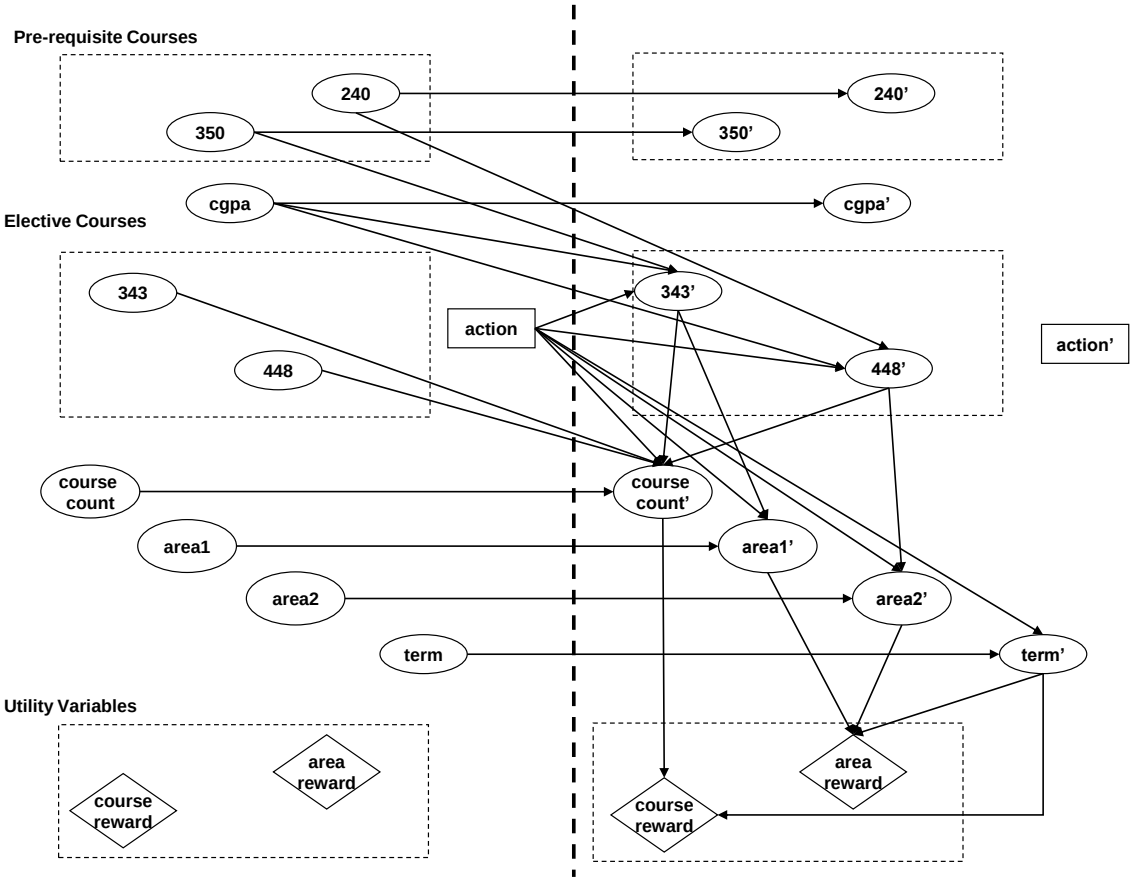


Figure 2.7: Dynamic decision network encoding of the course advising MDP

2.5.4 MDPs for Course Advising

The policy for a course advising MDP recommends the optimal course(s) in which a student should register in future terms. The reward function can encode various student preferences such as minimal time to graduate, maximize grade point average, or balance diversity vs specialization in choice of elective course.

A course advising MDP is used in Chapter 3 to demonstrate explanations generated for MDPs. The problem of course advising is modeled as a factored MDP that recommends elective courses to upper year students, based on their previous performance. The Dynamic Decision Network (DDN) associated with the MDP model is depicted in Figure 2.7. The assumption here is that the grades of a student in a new course can be predicted based on grades in pre-requisites and the cumulative grade point average (CGPA).

The model is based on a subset of the requirements at the David R. Cheriton School of Computer Science at the University of Waterloo. Undergraduate students must choose two elective courses in each of their last three terms, subject to fulfilling pre-requisite and course area constraints. Each course belongs to one of three areas. The grades are discretized and each course has a a grade variable taking one of four values $\{G, P, F, N\}$ corresponding to Good, Pass, Fail, and Not Taken. The default value is N. These variables correspond to the nodes in the recommender system in Figure 2.5.

In addition, book-keeping variables are used to store the CGPA (cumulative grade point average) with two possible values $\{G, P\}$, the total number of courses completed, whether each area has been covered, and the number of terms remaining. Each term corresponds to a level.

The action is choosing a pair of courses. The MDP policy depends on the values of all variables mentioned above, and recommends two courses to be taken in the next term. The transition function was constructed using historical data from University of Waterloo over several years (14,000 students). The reward function is decomposed additively into two components based on the degree requirements. Two utility variables, *course_reward* and *area_reward* are created that have high utility values for states in which 6 courses are passed and three areas are fulfilled. The rewards are only awarded at the end of the third term to avoid multiple accrual if a requirement is completed before the end of the last term. The objective is to maximize the values of these utility variables. Since the problem is finite-horizon the rewards are not discounted.

2.6 Summary

This chapter has formally introduced the framework of decision-theoretic planning. Decision-theoretic systems are based on probability theory and utility theory. Probability theory allows the explicit representation of uncertainty whether due to lack of sufficient knowledge or stochasticity in the environment. Utility theory allows the specification of user preferences. Both are then combined to model the problem of recommending an optimal action to a user, whether for a single-shot decision (modeled using Bayesian networks and influence diagrams) or in sequential planning (modeled as MDPs).

Real-world problems invariably result in exponential state space explosion and the solutions for optimal policies are computationally prohibitive. Factored models are often used to represent the state using a set of variables and also to compactly represent the transition function. Recommender MDPs, where the policy of the MDP is a recommendation to a user, can be modeled as factored MDPs. Two example recommender MDPs for

diagnostics and course advising are also introduced in this chapter. Course advising is used in Chapter 3 to demonstrate generation of explanations for MDP policies. Diagnostics is used in Chapter 4 to demonstrate refinement of Bayesian network model parameters and in Chapter 5 to demonstrate refinement of MDP model parameters.

Chapter 3

Explanations for MDPs

Often, a sequence of decisions must be taken by an individual or system. However, deciding on a course of action is notoriously difficult when there is uncertainty in the effects of the actions and the objectives are complex. Markov decision processes (MDPs) [79] provide a principled approach for automated planning under uncertainty. While such an automated approach harnesses the computational power of machines to optimize difficult sequential decision making tasks, the users no longer understand why certain actions are recommended. This lack of understanding is a serious bottleneck that is holding back the widespread use of MDPs as planning tools. Hence, there is a need for explanations that enhance the user's understanding and trust of these plans and help MDP designers to debug and validate their systems.

Merriam-Webster [1] defines explanation as the process of explaining. *To explain* is defined as to make known, or to make plain or understandable, or to give reason for or cause of, or to show the logical development or relationships. In terms of intelligent systems, explanation may be considered to be giving the reason or cause of the choice made by the system to make it understandable to the user. In terms of MDPs, this would mean demonstrating the process of computing the optimal policy for the MDP. The explanation of decision would require elaborating why a certain decision was chosen. It has been shown that explanations are needed either when the system performs an action that is not in accordance with the user's mental image [40], or when the cost of failure is high [88]. In this chapter, understanding and verifying the model that led to the optimal policy is the primary motivation.

Preliminary versions of parts of this chapter have been published in the 19th International Conference on Automated Planning and Scheduling (ICAPS) [52] and more recently

as a book chapter in a book on decision theory models [53].

3.1 Problem Statement

In MDPs, actions are selected according to the principle of maximum expected utility. Hence, explaining a decision amounts to explaining why the chosen action has highest expected utility. The expected utility depends on the probability of an event occurring and the utility of that event. Since MDPs deal with sequential planning, thus it is not intuitive to estimate the probability of an event occurring, possibly a few steps later. To address this issue, simple and easy-to-understand explanations need to be presented that provide insight into the expected utility computation by exposing some key pieces of information. More specifically, the frequencies of certain events that are more critical to the computation of the maximum expected utility are highlighted through explanation templates in this chapter. The focus of this work is not on natural language generation or sophisticated graphical user interfaces but on providing an intuition regarding the optimality of the policy by filling in pre-defined templates at run-time.

3.2 Literature Review

There has been less research on explanations in probabilistic and decision-theoretic systems compared to explanations for intelligent systems, such as expert and rule-based systems. This section first surveys the work on intelligent systems and then examines prior work on probabilistic and decision-theoretic systems.

3.2.1 Explanations for Intelligent Systems

Explanations for intelligent systems have been considered an essential component of such systems from their very outset. The primary focus of this previous work on explanations has been to provide users confidence regarding the recommendations made by the system.

Initial approaches to explanations in expert systems were based on presenting the execution traces to the users. This technique was used by the MYCIN [21] expert system for medical diagnosis. It provided users with answers to certain types of "Why" questions. They were answered by listing the rule that required certain information being asked from

the user as well as by providing a trace of the execution of that rule indicating the conclusion that rule could lead to. There have been other proposals to examine the code being run, and describe it in English as an execution trace [91].

The issue with providing an execution trace is that it can yield too much information, not necessarily all of which is relevant to understanding the final decision. Further, it is also not necessary that the structure of the expert system yields itself to user comprehension.

Another criticism of explanation systems based on execution traces has been that they do not provide justifications in their explanations. While the expert system can function and arrive at the correct conclusions by just knowing the rules, it is not possible to justify these conclusions without external domain knowledge which may be needed for certain domains. These kinds of explanations were needed especially for intelligent tutoring systems, in which the goal is to transfer knowledge to the user. Xplain [90] was one of the first systems to provide justifications of its decisions to users. To provide justifications, it maintained a knowledge base in addition to the rules used by the expert system. The knowledge base provided domain-specific information, called deep-knowledge by its designers, used for creating justifications.

In some domains other strategic information such as the ordering of those rules may also be needed to understand the choice of the best action. NEOMYCIN [21], an extension to MYCIN, addressed this problem by retaining strategic information of this nature, as well as requiring the designer to label different phases of the reasoning process explicitly. Examples of such phases could include establishing a hypothesis space, or exploring and refining that hypothesis space. The user was then given an explanation that included information about the high-level goals or sub-goals of the system relevant to the rule under consideration. In such a case, there is an additional burden on the designer of the system to record such information along with the rules or to explicitly divide the expert system's execution into such stages.

All the explanation techniques discussed above focus on exposing the line of reasoning of the expert system to the user. Wick and Thompson [100] argue that the best explanation may not necessarily be based on this line of reasoning. The reasoning process includes establishing potential hypotheses, then rejecting a few by considering additional information, and finally determining the strength of the best candidate hypothesis. Previous techniques would provide all of this information to the users, which might be too complex for them to understand. Thus, they advocate differentiating the line of explanation from the line of reasoning.

The line of explanation may be separated from the line of reasoning by making the explanation disregard the process used by the expert system. Given the conclusion of the expert

system, explanations may then be formulated by using additional domain knowledge[19]. Reconstructive explanations may not be reflective of the reasoning of the expert system, but at the same time they should not be factually incorrect. For this reason, such explanations are also called white lies [90].

Reconstructive explanations are not suitable for experts who are debugging the system. Their explanations must be based on the actual line of reasoning of the system for them to exactly understand how the system is behaving in a given configuration. Another issue with these explanations is the need for additional domain knowledge to construct the explanations.

Several systems also provide explanations based on similarity with other situations or cases known to the user. Quite a few recommender systems are based on collaborative filtering techniques in which users are grouped in different clusters based on their similarity on certain aspects. The system assumes that similar users have similar preferences. Thus, a user is recommended items (e.g., movies, books or songs) based on whether other users similar to her liked those items. Explanations for such recommendations [45, 94] convey how many users liked this item, or the attributes on which other users, who liked this item, were similar to the current user.

Case-based reasoning systems function on the principle of identifying other situations or events, called cases, that are most similar to the current case and then identifying the decisions taken in those cases. The users are then presented those cases as explanation for the chosen decision. Various similarity measures can be defined with the simplest being that the nearest neighbor is chosen as an explanation [2]. Another example of similarity can be the nearest neighbor that lies between the current situation and nearest unlike case, which in some cases can serve as a better explanation [28].

Similar cases have also been used as explanations for systems that use a complex mathematical technique, such as neural networks or regression [71]. In this case, the training data of the system is searched for a similar case with a similar result that can be presented to the user. Such explanations can be considered a form of reconstructive explanation, since the actual reason behind the choice is the mathematical technique rather than the similarity to the case in the training data.

Frame based systems [66] are not exactly a technique used to generate explanations, but they provide a similar functionality. Frames, originally proposed for computer vision and later also used in natural language processing, provide a method to represent information related to a specific situation. They provide high-level constructs that help identify the current state, and then low-level constructs, mirroring the properties of each high-level situation, to describe it. Slots are provided in frames to be populated at run-time, depend-

ing on different situations. This approach can be considered to provide different types of templates for different situations, that then need to be populated depending on the state of the system. Explanations for MDPs can be viewed similarly, where different types of explanations are needed for different situations, and templates need to be filled in with information about to explain the current situation.

3.2.2 Explanations for Decision-Theoretic Systems

While there has been a lot of work on explanations for intelligent systems, such as expert and rule-based systems, there has not been much work for probabilistic and decision-theoretic systems. The main reason behind this discrepancy is the difference in processes through which they arrive at their conclusions. For probabilistic and decision-theoretic systems, there are well-known axioms of probability that are applied to perform inference and theorems from utility theory that are used to compute a policy. Since these systems are based on a principled approach, experts do not need to examine the reasoning trace to determine if the inference or policy computation process is correct. The trace would essentially refer to concepts such as Bayes' theorem, or the principle of maximum expected utility, or other axioms of probability and utility theory. These techniques are well-known, and if it is assumed that they have been coded correctly, then there is no doubt that they will yield the correct result. This is in clear contrast with expert and rule-based systems. The experts need to examine what rules have been triggered as a result of the current state, and whether their sequence of instantiation and then execution is correct. This explains the large number of projects to generate explanations for expert and rule-based systems. On the other hand, for probabilistic and decision-theoretic systems, the requirement is to highlight portions of the input that lead to a particular result rather than to explain the principles behind these techniques. In the past, decision-theoretic approaches have not been scalable for real-world problems, which explains the lack of literature on explanations for MDPs and POMDPs. Decision-theoretic systems are now being proposed for use for relatively larger sized real-world problems [14, 67, 73], which now motivates the need for explanations in them. The use of larger sized real-world problems has also led to the proposal of approximate techniques for inference and policy generation to deal with scalability concerns. If the technique is approximate, then in such a case, it is again more important to provide an explanation to convince the expert that the approximation has not resulted in an incorrect solution.

The explanations of the previous section provide a good starting point for explaining MDPs, but do not serve the purpose completely. None of the systems discussed are stochastic, like MDPs. In most cases, the policy is already known to the designer, unlike MDPs in

which the policy is numerically computed. Also, none of them need to focus on a sequence of inter-related decisions, which is more complicated than explaining a single isolated decision. The complex numerical computations involved in finding optimal policies also make it difficult for users to gain an insight into the decision-making process. Druzdzel [30] rejected the notion that explanations for probabilistic systems cannot be generated due to these issues. In his work, causality was identified as a key component that could provide users with insight regarding the reasoning process. Since decision-theoretic systems (such as MDPs) are normative and not descriptive, it is even more important to provide users with explanations.

It may be noted here that a key issue in explanation of stochastic systems is the presentation of probability in different forms, such as numerical, verbal or graphical. An issue with using verbal explanations is that different people have different interpretations for terms such as "possibly", "probably", "likely" or "usually" [29].

Explanations in Bayesian Networks

Lacave et al. [56] provide a survey of different techniques in this area, and define three different types of explanations that are generated for Bayesian networks. The first type is related to explanation of evidence, in which the system explains what values of observed variables led to a certain conclusion. This technique is known as abduction. The second type is related to the explanation of the model in which the static knowledge, encoded in the Bayesian network, is presented or displayed to the user. The third technique refers to the explanation of reasoning in which the user is told how the inference process unfolds. This can be done by showing the reasoning process that led to certain results (possibly including intermediate results), or by showing the reasoning process that led to rejection of a hypothesis (possibly including intermediate results), or by providing knowledge to the user about hypothetical situations in which the result would have changed, if certain variables in the current state were different.

Chajewska and Helpburn [18] presented an approach for explanations in probabilistic systems by representing causality using Bayesian Networks and exploiting different links. The approach presented in this chapter is similar as it also uses templates to generate explanations and analyze the effects of the optimal action. However, the analysis is not restricted to a single relevant variable and the the long-term effects of the optimal action (beyond one time step) are also considered.

Explanations in Influence Diagrams

Lacave et al. [57, 58] present different approaches to explain graphical models, including Bayesian networks and influence diagrams. Their explanations are geared to users with a background in decision analysis, and they present utilities of different actions graphically and numerically. This work can be used to assist experts, and the authors mention that they have used it to construct and debug models to help medical doctors in diagnosis and decision-making. Similar techniques may be used to debug and validate the reward function for an MDP, but not the transition function. Furthermore, there is still a need to consider the effect of sequential decision-making.

Explanations in MDPs

There has been very little research on explanation of policies generated using MDPs with the exception of two other streams of work, that of Elizalde et al. [33, 32, 34] which was formulated at the same time as the work in this chapter and the work by Dodson et al., [25, 26] that was published subsequent to the publication of the work in this chapter.

In the work of Elizalde et al., [33, 32, 34] an explanation comprises three components: an optimal action, a relevant variable, and explaining why the optimal action is the best in terms of the relevant variable. They identify the relevant variable by determining which variable affects the utility function the most. Two heuristics are provided to determine a relevant variable. In the first method, they keep the rest of the state fixed and only change the values of one variable under consideration. By doing this, they measure the difference in the maximum and minimum values of the utilities of the states which are similar for all other variables except for the variable being considered. This process is repeated for all variables, and the variable with the largest difference between the maximum and minimum value is then considered relevant. In the second heuristic, they examine the optimal action for different states, such that only the value of the variable under consideration is changed and other values are kept fixed. They consider a variable more relevant if the optimal policy changes more frequently by changing the value of that variable, while keeping the values of other variables fixed. Such explanations belong to the category of reconstructive explanations since they do not mirror the reasoning process. They provide users intuition regarding the optimal action, by discussing the relevant variable and its possible impact but may not be useful for debugging purposes. It is also conceivable that multiple relevant variables may need to be combined to construct a more meaningful and intuitive explanation. It will be interesting to explore the long-term effects of the optimal action, rather than only focusing on the change of utility while identifying a relevant variable.

Subsequent to the publication of the work in this chapter [52], Dodson et al., [25, 26] have also described an approach based on natural language argumentation for explanations of MDP policies. Coincidentally, they also use the domain of academic course advising [41] similar to the evaluation domain used in this chapter. The technique described by Dodson et al., [25, 26] is more focused in helping end-users, who may not be very knowledgeable about the concept of an MDP or comfortable with the use of probabilities, to understand the explanations. Thus, the goal is not essentially to provide the minimum information required to prove the optimality of the policy but to convince an average user to trust the policy. In line with this goal, they also focus on presenting the explanations using a natural language interface. They also conduct a user study to compare their explanations with those presented in this thesis and demonstrate that users are more accepting of explanations provided in their natural language interface. The focus of the work in this chapter is to help experts understand the model and determine if any corrections are necessary to the transition function.

McGuinness et al. [64] identify several templates to present explanations in task processing systems based on predefined workflows. The approach in this chapter also uses templates, but predefined workflows cannot be used due to the probabilistic nature of MDPs.

3.3 Explanations for MDPs

The objective of generating an explanation of an MDP policy is to answer the question,"*Why has this recommendation been made?*". The technique presented in this section populates generic templates at run-time, with a subset of those included in the explanation.

3.3.1 Templates for Explanations

The reward function reflects the preference amongst different states or scenarios. Rewards are generally assigned to states or scenarios that have certain semantic value associated with them. The policy for an MDP is computed by maximizing the sum of expected discounted rewards (Equation 2.7). The explanations should indicate how this expectation is being maximized by executing the optimal action. The approach in this section anticipates the effects of an action and shows the contributions of those effects to the sum of expected rewards.

An alternate method to evaluate a policy involving occupancy frequencies [79] is used in generating the explanations. The discounted occupancy frequency (hereafter referred as simply occupancy frequency), $\lambda^{\pi}_{s_0}(s')$, is the expected number of times one will reach state s' from starting state s_0 by executing policy π. Occupancy frequencies can be computed by solving Equation 3.1.

$$\lambda^{\pi}_{s_0}(s') = \delta(s', s_0) + \gamma \sum_{s \in S} T(s', \pi(s), s)\, \lambda^{\pi}_{s_0}(s) \quad \forall s' \tag{3.1}$$

where $\delta(s', s_0)$ is a Kroenecker delta which assigns 1 when $s' = s_0$ and 0 otherwise. The occupancy frequency is not a probability so it can lie in $[0, h]$. In domains where it is impossible to revisit a state, occupancy frequency will lie in $[0, 1]$ and can be considered as a probability. The dot product of occupancy frequencies and rewards gives the value of a policy, as shown in Equation 3.2.

$$V^{\pi}(s_0) = \sum_{s \in S} \lambda^{\pi}_{s_0}(s)\, \rho(s, \pi(s)) \tag{3.2}$$

The value of the policy is the sum of products of the occupancy frequency of each state with its reward. An explanation for choosing an action could be the frequency of reaching a state is highest (or lowest) relative to other actions. This is especially useful when this state also has a relatively high (or low) reward. Below are a list of templates in which the underlined phrases (states and their probabilities) are populated at run-time.

- **Template 1:**"*ActionName* is the only action that is likely to take you to $State_1$ about λ times, which is higher (or lower) than any other action"
- **Template 2:** "*ActionName* is likely to take you to $State_1$ about λ times, which is as high (or low) as any other action"
- **Template 3:** "*ActionName* is likely to take you to $State_1$ about λ times"

As mentioned earlier, the frequency λ can be higher than 1 if a state can be revisited. While the frequencies are discounted they still represent an expectation of the number of times a state will be visited. To understand this, consider an alternate yet equivalent representation of the discount factor in which $1 - \gamma$ is the termination probability, *i.e.*, the probability of the MDP terminating at each step. For problems, without any discount factor, the frequencies will not be discounted. In general, all MDPs, including discounted

problems, can be converted to stochastic shortest path MDPs [63] without any discount factor and an indefinite horizon. The discount factor is retained in all equations in this thesis, but it can be ignored without the loss of generality, if desired.

The above templates provide a method to present explanations but multiple templates can be populated even for non-optimal actions; a non-optimal action may have the highest frequency of reaching a state with a high reward, but it still may not have the maximum expected utility. Thus, a process is needed to identify a set of templates to include in the explanation to justify the optimal action.

3.3.2 Minimum Sufficient Explanations (MSE) for Flat MDPs

In this chapter, the concepts of minimum and sufficient explanations are introduced. An explanation is defined as sufficient if it can prove that the recommendation is optimal, *i.e.*, the selected templates show the action is optimal without needing additional templates. A sufficient explanation cannot be generated for a non-optimal action since an explanation for another action (*i.e.*, optimal action) will have a higher utility. A sufficient explanation is also defined as minimum if it includes the minimum number of templates needed to ensure it is sufficient. The sufficiency constraint is useful in trying to debug the model. The minimum constraint is useful in trying to understand the policy with as little information as is necessary. If needed, either of the minimum or sufficiency constraints can be relaxed to present more ore less templates depending upon the audience and purpose of the explanation.

Let s_0 be the state where additional information is desired to explain why $\pi^*(s_0)$ is an optimal action. The value of executing any action at state s_0 can be calculated using the Q-function defined in Equations 2.8. The Q-function can also be evaluated using occupancy frequencies as defined in Equation 3.2. Since a template is populated by a frequency and a state, the concept of a term t is introduced here to encapsulate this information as $t(s, \pi^*, s_0) = \lambda^{\pi^*}_{s_0}(s)\,\rho(s, \pi^*(s))$. Now V^{π^*} can be computed using Equation 3.3.

$$V^{\pi^*} = \sum_{s \in S} t(s, \pi^*, s_0) \tag{3.3}$$

The minimum sufficient explanation (MSE), defined above, comprises a subset of the terms in Equation 3.3. For this reason, the expected utility of the terms included in the MSE, V_{MSE}, cannot exceed the value of the optimal action, V^{π^*}, but it must also be higher than the value of any other action, *i.e.*, $V^{\pi^*} \geq V_{MSE} > Q^{\pi^*}(s_0, a) \quad \forall a \neq \pi^*(s_0)$. In the

worst case, all terms will have to be included in the explanation. To compute the MSE, all terms in Equation 3.3 can be arranged in descending order, and then the first k terms of this sequence, necessary to ensure that $V_{MSE} \geq Q^{\pi^*}(s_0, a)$, can be selected. The value of an MSE, V_{MSE}, is formally defined using Equation 3.4.

$$V_{MSE} = \sum_{i \leq k} t_i + \sum_{i > k} \lambda_{s_0}^{\pi^*}(s_i)\, \overline{r} \tag{3.4}$$

V_{MSE} in Equation 3.4 comprises two components. First, the expected utility from all the terms in the MSE, *i.e.*, $\sum_{i \leq k} t_i$. Second, for every term not included in the MSE, its worst case is assumed by adding utility computed by using the minimum possible reward, $\overline{r}$, to the MSE. The second component is needed to ensure sufficiency if rewards are negative, and minimum otherwise.

3.3.3 Minimum Sufficient Explanations (MSE) for Factored MDPs

In Equation 3.3, the total number of terms will equal the size of the state space. This can be computationally prohibitive for large state spaces. Typically, factored MDPs are used in such cases. Occupancy frequencies for scenarios, $\lambda_{s_0}^{\pi}(sc)$, in factored MDPs can be defined as the expected number of times one will reach a scenario sc, from starting state s_0, by executing policy π *i.e.*, $\lambda_{s_0}^{\pi}(sc) = \sum_{s \in sc} \lambda_{s_0}^{\pi}(s)$. Also, the reward function can be defined by a set $\mathcal{R}$ of reward variables R such that the sum of their values r is the reward at any given state (e.g. $\rho(s, a) = \sum_{R \in \mathcal{R}} r_{R,s,a}$, where $r_{R,s,a}$ represents the reward for reward variable R in state s of executing action a). Let $sc_{R=r}$ define the scenario for which reward variable R has value r[1], and $V_f^{\pi^*}$ represent the utility of executing π^* for a factored MDP. In Eq. 3.4, $\overline{r}$ represents the minimum value for the reward function. With multiple reward variables, every variable may have its own minimum value which can be used instead. Let $\overline{r}_i$ define the minimum value for the reward variable used in term i in the sorted sequence. Now, Equations 3.3 and 3.4 can be re-written as Equations 3.5 and 3.6 respectively.

$$V_f^{\pi^*} = \sum_{R \in \mathcal{R}} \sum_{r \in dom(R)} \lambda_{s_0}^{\pi^*}(sc_{R=r})\, r \tag{3.5}$$

[1]For the sake of completeness, it should be pointed out that the special case where a set of scenarios for reward variable R have value r can also be handled by computing the occupancy frequency for each scenario independently and then adding them to create a single term for use in Equation 3.5 or otherwise retain a separate term for each such scenario.

$$V_{fMSE} = \sum_{i \leq k} t_i + \sum_{i > k} \lambda_{s_0}^{\pi^*}(sc_i)\,\overline{r}_i \tag{3.6}$$

The number of terms is now significantly lower since only a single term per value of each reward variable is needed. This allows computing an MSE even for domains with large state spaces. Additionally, the templates can also be modified for the factored case as listed below:

- **Template 1:**"*ActionName* is the only action that is likely to take you to $Var_1 = Val_1, Var_2 = Val_2, ...$ about $\underline{\lambda}$ times, which is higher (or lower) than any other action"
- **Template 2:** "*ActionName* is likely to take you to $Var_1 = Val_1, Var_2 = Val_2, ...$ about $\underline{\lambda}$ times, which is as high (or low) as any other action"
- **Template 3:** "*ActionName* is likely to take you to $Var_1 = Val_1, Var_2 = Val_2, ...$ about $\underline{\lambda}$ times"

It is known that the optimal policy is invariant to positive linear transformations on the reward function [97]. This property in the MSE is also desired to ensure that the MSE only changes if the model has changed. This will assist designers in debugging the model efficiently.

Proposition 1. *MSE remains invariant under affine transformations of the reward function.*

Proof. Let $\check{V}_f^{\pi}$ denote the expected utility for any policy π when rewards have been scaled by adding any constant c. If r is substituted by $r + c$ in Eq. 3.5 it can be rewritten as $\check{V}_f^{\pi} = V_f^{\pi} + c \sum_{R \in \mathcal{R}} \sum_{r \in dom(R)} \lambda_{s_0}^{\pi}(sc_{R=r})\, r$. Since occupancy frequencies computed for an MDP must add up to the horizon ($\sum_{r \in dom(R)} \lambda^{\pi}(sc_{R=r})\, r = h$), so $\check{V}_f^{\pi} = V_f^{\pi} + c|\mathcal{R}|h$, where $|\mathcal{R}|$ is the total number of reward variables. Similarly $\check{V}_{fMSE} = V_{fMSE} + c|\mathcal{R}|h$. Since $V_{fMSE} > V_f^{\pi}$, thus $\check{V}_{fMSE} > \check{V}_f^{\pi}$ for any constant c. Also $\check{V}_{fMSE}$ will comprise the same scaled reward values and frequencies as those in V_{fMSE} otherwise the explanation would not remain either sufficient or minimum. For discounted domains, the frequencies will add up to the expected discounted horizon instead of h. A similar proof can also be presented for the case where rewards are multiplied by a positive constant. □

3.3.4 Workflow and Algorithm

The basic workflow for the explanation process is as follows. The designer identifies the states and actions, and specifies the transition and reward functions of an MDP. The optimal policy is computed by using a technique such as value iteration. Now the designer can consult an optimal policy to determine an optimal action and request an explanation. The explanation is generated using the process listed below:

1. Compute $sc_{R=r}$, the scenario which comprises the set of all states that lead to each value r of each reward variable R. This information is directly available from the dependencies encoded in the reward function. For each partial assignment of value r to variable R, note the set of states that receive reward r to compute the scenario.

2. For every scenario $sc_{R=r}$, compute the occupancy frequency $\lambda_{s_0}^{\pi^*}(sc_{R=r})$ for every action using Eq. 3.1. The occupancy frequency for a scenario is computed efficiently by summing the occupancy frequencies of each state in it using variable elimination. The recurrence is terminated after a number of steps equal to the horizon of the MDP or when convergence is achieved (due to the goal state in indefinite horizon problems or otherwise the discount factor) for infinite horizon problems.

3. Compute the term $t(sc_i, \pi^*, s_0)$ and $\lambda_{s_0}^{\pi^*}(sc_i)\,\overline{r}_i$ for every scenario $sc_{R=r}$. They respectively represent the advantage and disadvantage of including and excluding a term from the MSE.

4. Sort $t_i - \lambda_{s_0}^{\pi^*}(sc_i)\,\overline{r}_i$ in descending order and select the first k terms from this sequence to include in the MSE for which $V_{MSE} > Q^{\pi^*}(s_0, a) \quad \forall a \neq \pi^*(s_0)$. Note that t_i and $\lambda_{s_0}^{\pi^*}(sc_i)\,\overline{r}_i$ respectively represent the advantage and disadvantage associated with including and excluding a term from the MSE, so their difference indicates the benefit of this term in the explanation versus excluding it.

5. Present each term in the explanation to the user in one of the defined templates. Choose templates using the following criteria.

   (a) Use template 1 if the optimal action has the highest (or lowest) expected frequency to reach that scenario by a significant margin[2].

   (b) Use template 2 if the optimal action has the highest (or lowest) expected frequency, but not by a significant margin.

[2]In the implementation for experiments in this chapter, a significant margin means twice as high as the next highest. It can be adjusted depending on the domain.

(c) Use template 3 if neither of the previous templates can be used.

The above process to compute MSE can be summarized as Algorithm 1.

Algorithm 1 Computing Minimum Sufficient Explanations at state s_0 using policy π^*

computeMSE(s_0,$\pi*$)

for each r in R
  $sc[r] \leftarrow$ computeScenarios(r)
  λ[r] $\leftarrow$ computeOccupancyFrequency($sc[r]$)
  $utilTemplate[sc, \pi^*, s_0] \leftarrow \lambda[r] \cdot r$
  $utilNoTemplate[sc, \pi^a, s_0] \leftarrow \lambda[r] \cdot \bar{r}$
  $netUtil[sc, r] \leftarrow utilTemplate[sc, \pi^*, s_0] - utilNoTemplate[sc, \pi^a, s_0]$
$sortedNetUtil \leftarrow$ sortDescending($netUtil$)
Initialize $V_{MSE} \leftarrow 0$, $V_{Templates} \leftarrow 0$ and $V_{NoTemplates} \leftarrow \sum utilNoTemplate$
Initialize $k \leftarrow 0$, $MSE \leftarrow \{\}$ and $Pairs \leftarrow \{\}$
repeat
    add $sc[r], \lambda[r]$ for $sortedNetUtil[k]$ in $Pairs$
    $V_{Templates} \leftarrow V_{Templates} + utilTemplate[k]$
    $V_{NoTemplates} \leftarrow V_{NoTemplates} - sortedNetUtil[k]$
    $V_{MSE} \leftarrow V_{Templates} + V_{NoTemplates}$
    $k \leftarrow k + 1$
until ($V_{MSE} < V_{next}$)
$MSE \leftarrow$ generateTemplates($Pairs$)
return MSE

The function computeScenarios returns the set of scenarios with reward value r which is available in the encoding of the reward function. The function computeOccupancyFrequency is the most expensive step which corresponds to solving the system of linear equations defined in Equation 3.1, which has a worst case complexity that is cubic in the size of the state space. However, in practice, the running time can often be sublinear by using variable elimination [103] to exploit conditional independence and algebraic decision diagrams [46] to automatically aggregate states with identical values/frequencies. The variable V_{next} refers to the value function of the state with the second highest value. The function generateTemplates chooses an applicable template, from the list of templates, in the order of the list, with the last always applicable.

For any given state, an MSE is guaranteed to exist; in the worst case it will need to include all the terms in the MSE from Eq. 3.3 or Eq. 3.5. Thus, the upper bound on the number of templates displayed to the user is also given by the number of terms, which will depend on the structure of the MDP being explained. A relatively large number of terms in the MSE will indicate that the effect of the optimal action is not substantially different from that of at least one other action. It can also be argued that for every term there is at least one template that can be used to present the information to the user since Template 3 can always be used. While template 3 may not seem to provide much information in itself, it does indicate that there are better or worse actions available if the scenario being depicted is of particular interest to the user or designer.

3.4 Experiments and Evaluation

The approach to generate MSE was evaluated by running experiments on two different domains: course advising and handwashing. The course-advising MDP has 4 core courses and 7 elective courses (from which the student has to choose), with each course having 4 possible letter grades and belonging to a certain area. It has 21 possible actions, with each action representing a pair of elective courses. The objective is to pass 6 elective courses in 3 terms by taking at least one course in 3 different areas. The transition model was obtained by using historical data collected over several years (for 14,000 undergraduate students) at the University of Waterloo. The reward function provides rewards for completing different degree requirements with the reward function decomposed in two different variables, one for each degree requirement, with 2 values per variable. The horizon of this problem is 3 steps, each step representing one term and it is undiscounted. The other domain was the handwashing POMDP developed by Hoey et al. [47], available online, to assist people with dementia in handwashing. The POMDP was converted into an MDP, assuming all states are observable and changing variable names and values to make them more user-friendly. The horizon for this domain is 100 steps and discount factor is 0.95. There are three different reward variables in the reward function with 19 distinct values for rewards. Explanations for this domain are intended for a caregiver/nurse to evaluate the validity of the prompt, and not for the person washing hands.

MSEs were calculated for different starting states in the course advising and handwashing MDPs. The results are shown in Table 3.1 and Table 3.2. It can be seen that the MSE generally contains very few terms for both domains, more evident in the handwashing domain in which there was a total of 19 terms, and only 6 were needed for any given optimal action in 382 different starting states generated randomly. It is natural to

Table 3.1: Explanations for Course Advising Domain (Reward Variables=2, Values per Variable=2+2, Max. Terms=4)

Terms in MSE	1	2	3–4
Frequency	134	48	0
Mean ±STD of $Q^{\pi^*}(s_0, a')/V^{\pi^*}$	0.46±0.41	0.81±0.24	-

Table 3.2: Explanations for Handwashing Domain (Reward Variables=3, Values per Variable=2+2+15, Max. Terms=19)

Terms in MSE	1	2	3	4	5	6
Frequency	0	142	94	119	2	25
Mean ±STD of $Q^{\pi^*}(s_0, a')/V^{\pi^*}$	-	0.51±0.22	0.62±0.10	0.68±0.04	0.61±0.15	0.69±0.05

expect an explanation to be more complicated if two policies have similar effects. The complexity of an explanation is estimated by the number of terms included in it. Also two policies are considered to have similar effects if the ratio of their normalized expected values is close to 1. It can be seen from both tables that if the ratio between the expected utility of the second best policy, $Q^{\pi^*}(s_0, a')$, and optimal policies, V^{π^*}, is high then the explanation includes more terms. On the other hand, if the ratio is low it means that the optimal action is much superior and we can see that fewer terms are needed in the MSE. This result is intuitive as more templates would be expected in the MSE if the optimal policy is pretty similar to another policy.

Two sample explanations, one from each domain, are shown below.

- Action *TakeCS343&CS448* is the best action because:-
  - It is likely to take you to $CoursesCompleted = 6$, $TermNumber = Final$ about 0.86 times, which is as high as any other action
- Action *DoNothingNow* is the best because:-
  - It is likely to take you to $handswashed = YES$, $planstep = Clean\&Dry$ about 0.71 times, which is the higher than any other action
  - It is likely to take you to $prompt = NoPrompt$ about 12.71 times, which is as high as any other action[3]

[3]The variable names have been changed from those in [47]

The occupancy frequency in the last template is higher than 1 because it is possible visit a state multiple times with a reward for not issuing a prompt every time.

The optimal policies for both domains were pre-computed since they do not need to be recomputed for every explanation. It took approximately 1 and 4 seconds respectively to generate explanations for the course advising and handwashing problems on a Pentium IV 1.66 GHz laptop with 1GB RAM using Java on Windows XP with the optimal policy and second best action precomputed. Note that the course advising problem has 117.4 million states and the handwashing problem has 207,360 states. The simpler reward function (two reward variables with two values each) in the course advising domain resulted in faster execution despite a larger number of states.

A user study was conducted to evaluate explanations for course advising with advisors and students. The results of this study are presented next.

3.4.1 Feedback from Advisors

The explanations were presented to the advisors for several states to undergraduate advisors and to seek their feedback. The advisors noted that students not performing well would benefit from these explanations as they would help them focus on requirements they are likely to not fulfill. They also mentioned that grade-conscious students would also appreciate such explanations. The advisors considered the model used by the MDP, *i.e.*, the transition probabilities, as a useful tool to validate or correct their perception about various courses being easy or hard. They were apprehensive that it would be difficult for an online system to replace them as they consider more factors than completing degree requirements, which include preferences such as student's interests, career path, difficulty level of the course, stereo-types about areas, courses or professors. There has been some research on preference elicitation to model student preferences for course advising in MDPs [83], so it was explained to them that it is possible to extend the current course-advising model to include such factors, but that is outside the scope of this work on explaining MDPs.

3.4.2 User Study with Students

A total of 37 students were recruited from the University of Waterloo's Cheriton School of Computer Science. These students were shown 5 recommendations with explanations for different states. For each explanation, they were asked to rate the explanations on various factors such as comprehensibility, trust-worthiness and usefulness. For 3 states,

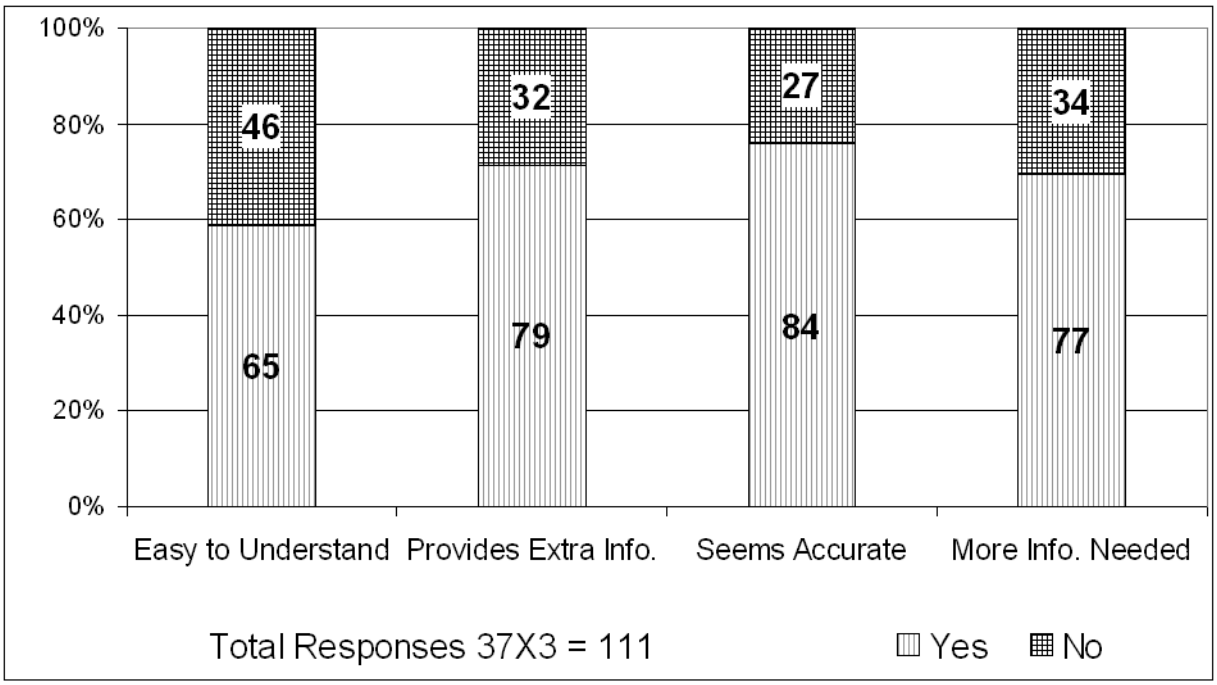


Figure 3.1: User perception of MDP explanations

the explanation was computed by our technique and for the other 2, the explanations were provided by advisors.

The results regarding the user's perceptions of these explanations are shown in Figure 3.1. 59% (65/111) of the respondents indicated that they were able to understand the explanation without any other information. Most of the students who wanted more information either wanted to know the occupancy frequencies for some other actions to get a comparison, or more knowledge about the technique used to compute the transition probabilities. For the first concern, this information can be provided as it is already computed in Step 2 of algorithm for the MSE. For the second concern, this curiosity can be attributed to the audience mostly being students in Computer Science who are interested in understanding the system.

In 76% (84/111) of the cases, it was believed that the explanation provided by the system was accurate. A few students wanted to know the sample size of the data used to learn the transition function to judge the accuracy. Quite a few respondents, 69% (77/111), indicated that they would require extra information beyond that presented in the explanation. When asked what other type of information they may be interested in,

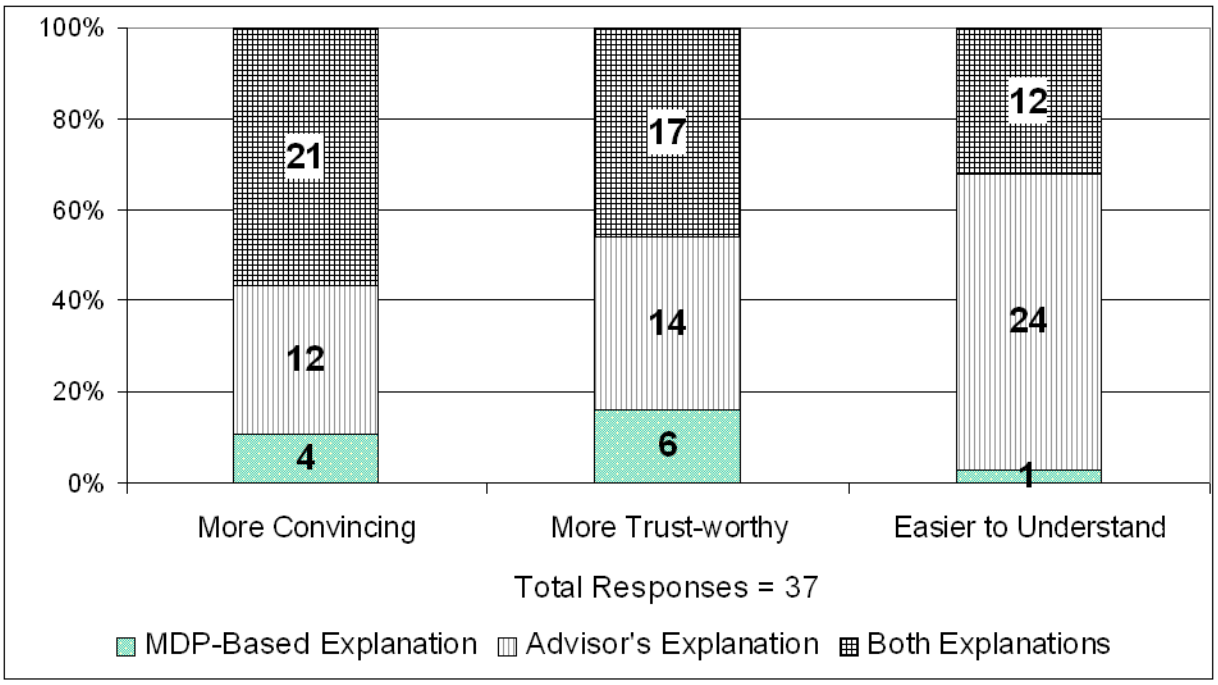


Figure 3.2: Comparison of MDP and Advisor explanations

it was revealed that they considered the model inadequate and wanted a richer model rather than the explanation being inadequate for the existing model. The requests for the richer model included the ability to cater to preferences such as student's interest, future career plans, and level of difficulty. An important indicator of the usefulness of these explanations is that 71% (79/111) of the students mentioned that the explanation provided them with extra information that helped them in making a decision. Another indicator of their influence is that while students initially disagreed 23% (26/111) of the times with the recommendation, in 35% (9/26) of these cases the explanation convinced them to change their mind and agree with the original recommendation. In most of the other cases, again the students disagreed because their final decision depended upon some factor, not modeled by the system, so their opinion could not have been changed by any explanation without first enriching the model further.

To compare these explanations with advisor explanations, the students were asked whether they preferred these automatically generated explanations, the advisor explanation, or having access to both of them simultaneously. These results are shown in Figure 3.2. 57% (21/37) students found that the most convincing option was to access both explana-

tions, as opposed to 32% in favor of only advisor explanations and 11% in favor of only our explanations. Similarly, 46% (17/37) students considered both explanations viewed together as more trustworthy as opposed to having only advisor explanations (38%) or only our explanations (16%). As expected, most of the students (65% or 24/37) found it easier to understand advisor explanations as they were more personal and human-oriented. This is understandable since the advisors employ domain-specific constructs in their explanations while our explanations are totally generic. However, a few students (32%) also indicated that having a combination of the two would be easier to understand. Finally, the students were also asked if they were provided access to such a system over the Internet, in addition to the option of discussing their choices with an undergraduate advisor, would they use this system. 86% of them mentioned they would use it from home while trying to determine their choice of courses, 89% mentioned they would use it before meeting with an advisor to examine different options for themselves, and 70% mentioned they would use it after meeting with advisors to arrive at a final decision. Among the 30%, who indicated they would not use it after meeting advisors, many expected the advisors to incorporate the information from the automatically generated explanations in their advice and thus considered it redundant to check it themselves. In any case, these numbers are very encouraging as they show substantial interest in the explanations.

The explanations generated by the system are generic, while those provided by the advisors are domain-specific. The results show that these two types of explanations are complementary and students would like to access these explanations in addition to consulting advisors. A recurring theme during the user study was students inquiring about a facility to compare different choices, *i.e.*, asking the question "*Why is action a better than action b*?", especially if their preferred action was different from the recommended action. The system has now been extended to answer such questions by presenting the MSE for the optimal action π^* and then populating the templates for the same terms for the action users want to compare against action b. The comparison demonstrates how action a is better than action b as $V_{MSE} \geq Q^{\pi^*}(s_0, b)$.

3.5 Summary

Transition functions for MDPs are often learnt from data which may not be sufficient or otherwise representative of the actual problem being modeled. It is natural for human beings to question the rationale behind a decision which is being recommended for a multi-step problem in an environment with stochastic effects. Thus, MDP policies need to be explained when questioned by human beings. This chapter presents a mechanism to

generate explanations for flat and factored MDPs in any domain without additional effort from the MDP designer. The explanations are presented in easy to understand templates that indicate the possible consequence of executing an action in terms of states and scenarios that are interesting for the reward function. The concept of a minimum sufficient explanation is also introduced through which an action can be explained using the fewest possible terms. The use of additional terms in the explanation is superfluous as they would not effect the optimality of an action.

The technique to generate explanations is demonstrated on two real-world domains, i.e., course-advising and handwashing. The results of a user study that evaluates the effectiveness of these explanations for course advising are also presented. It is shown that users appreciate the extra information provided by the generic explanation calculated through the approach described in this chapter. However, the users also require domain-specific information in the explanations to completely convince them. Most of the students who participated in the user study considered the combination of MDP explanations along side human advisor explanations as more effective than either one alone.

The technique for generating explanations presented in this chapter is agnostic to the particular technique used to solve MDP policies. In particular, the implementation in this chapter was based on an approach for solving MDPs based on Algebraic Decision Diagrams (ADDs) [46] to demonstrate the solution on large sized real-world problems. Other techniques, including approximate techniques, can also be used for the purpose of solving MDPs and then computing the explanations without any loss of generality.

Explanations for MDPs presented in this chapter are generated with the assumption of conveying information to a rational decision-maker. The objective here is to ensure that a human being is provided the correct information required to understand the choice of the system which the system believes will maximize the expected utility. Thus, the goal of explanations is more oriented towards understanding if the model is correct. If the primary purpose of the explanation is to gain user acceptance, then it is possible to generate explanations that exploit various cognitive biases that human beings are susceptible to [86, 92]. For instance, explanations can include the notion of confidence of the system to convey a sense of security, explanations can compare the choice with more extreme cases to present the choice in a favorable fashion, non-relevant information may be added to an explanation to sway the user, or the explanation can be framed in terms of positive concepts (reward function is rescaled such that less favorable situations have low positive reward rather than a negative reward). Similarly, instead of generating the minimum sufficient explanation, the templates can also be chosen based on what may be most likely accepted by the user.

Explanations can help convince users but they will not be useful if the model through

which the policy is computed is incorrect. In such a situation, the user will want to execute an alternate action. The next two chapters discuss techniques to refine models for Bayesian network and MDPs based on expert feedback.

Chapter 4

Refining Models for Bayesian Networks

Consider the problem of diagnostics where a diagnostic Bayesian network is available. This Bayesian network can be used to infer a distribution over causes given some available evidence in the form of observed tests. If the decision-making is being performed by an expert, the natural objective would be to select the test that will yield the highest reduction in uncertainty about the underlying cause. So if an expert can be observed performing some tests, with the assumption that the tests are being selected optimally, then the Bayesian network model should also be consistent with the expert. The consistency can be evaluated by determining the test that provides the highest expected reduction in uncertainty and comparing it against the expert's choice. This chapter examines this problem.

A preliminary version of parts of this chapter have been published in the proceedings for the 2011 conference on Neural Information Processing Systems (NIPS) [50].

4.1 Problem Statement

A model that is consistent with an expert would generate Gini impurity rankings (as defined in Equation 2.4) consistent with the expert's diagnostic sequence. This is because the Gini impurity reduces the expected uncertainty and the expert is also assumed to be working with the objective of selecting the test that would maximize the reduction in uncertainty over the causes. The expert's test choices are thus interpreted as implying constraints on Gini impurity rankings between tests. To that effect,Agosta et al., [6] define the notion

of *Cause Consistency* and *Test Consistency*, which indicate whether the cause and test orderings induced by the posterior distribution over causes and the VOI of each test agree with an expert's observed choice. Assuming that the expert greedily chooses the most informative test T^* (i.e., test that yields the lowest Gini impurity) at each step, then the model is consistent with the expert's choices when the following equations are satisfied for each constraint:

$$\mathrm{GI}(\mathbf{C}|T^*) \leq \mathrm{GI}(\mathbf{C}|T_i) \qquad \forall i \tag{4.1}$$

The objective of refining the parameters of the MDP is to ensure that all constraints learnt from the experts are satisfied using the refined parameters.

4.2 Literature Review

Parameter learning for Bayesian networks can be viewed as searching in a high-dimensional space. Adopting constraints on the parameters based on some domain knowledge is a way of pruning this search space and learning the parameters more efficiently, both in terms of data needed and time required. Qualitative probabilistic networks [98] allow qualitative constraints on the parameter space to be specified by experts. For instance, the influence of one variable on another, or the combined influence of multiple variables on another variable [31] leads to linear inequalities on the parameters. Wittig and Jameson [101] explain how to transform the likelihood of violating qualitative constraints into a penalty term to adjust maximum likelihood, which allows gradient ascent and Expectation Maximization (EM) to take into account linear qualitative constraints.

Other examples of qualitative constraints include some parameters being larger than others, bounded in a range, within ϵ of each other, etc. Various proposals have been made that exploit such constraints. Altendorf et al. [8] provide an approximate technique based on constrained convex optimization for parameter learning. Niculescu et al. [69] also provide a technique based on constrained optimization with closed form solutions for different classes of constraints. Feelders [35] provides an alternate method based on isotonic regression while Liao and Ji [60] combine gradient descent with EM. de Campos and Ji [22] also use constrained convex optimization, however, they use Dirichlet priors on the parameters to incorporate any additional knowledge. Mao and Lebanon [62] also use Dirichlet priors, but they use probabilistic constraints to allow inaccuracies in the specification of the constraints.

A major difference between the technique presented in this chapter and the previous work is on the type of constraints. The constraints discussed in this chapter do not need to

be explicitly specified by an expert. Instead, passively observing the expert and learning from what choices are made and not made [74] allow the learning of these constraints. Furthermore, as will be shown later, the constraints based on expert feedback are non-convex, preventing the direct application of existing techniques that assume linear or convex functions. In this chapter, Beta priors are used on the parameters, which can easily be extended to Dirichlet priors for cases where the causes are not Boolean variables. The constraints are incorporated in an augmented Bayesian network, similar to Liang et al. [59], though their constraints are on model predictions as opposed to these which are on the parameters of the network. Finally, this chapter also uses the notion of probabilistic constraints to handle potential mistakes made by experts.

4.3 Model Refinement for Bayesian Networks

A recent approach for easing the bottleneck of knowledge acquisition grew out of the realization that the best time to gain an expert's insight into the model structure is during the diagnostic process. Recent work in "Query-Based Diagnostics" [6] demonstrated a way to improve model quality by merging model use and model building into a single process. More precisely the expert can take steps to modify the network structure to add or remove nodes or links, interspersed within the diagnostic sequence. In this chapter, this variety of learning-by-example is extended to include refinement of model parameters based on the expert's choice of test, from which constraints are determined. The nature of these constraints defined through expert feedback is derived from the value of the tests to distinguish causes, so Gini impurity can be used to evaluate them.

The example of diagnostic Bayesian networks is used in this chapter but the techniques mentioned here can be applied for other similar recommender Bayesian networks for other domains.

4.3.1 Augmented Diagnostic Bayesian Network

Consider a simple diagnosis example with two possible causes C_1 and C_2 and two tests T_1 and T_2 as shown in Figure 4.1. To keep the exposition simple, suppose that the priors for each cause are known (generally separate data is available to estimate these), but the conditional distribution of each test is unknown[1]. Using the Noisy-OR parameterizations

[1]The technique presented in this chapter can also be used for the case where the priors over causes are not known

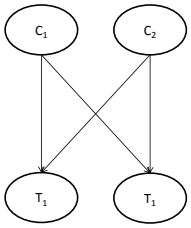


Figure 4.1: Diagnostic Bayesian network with 2 causes and 2 tests

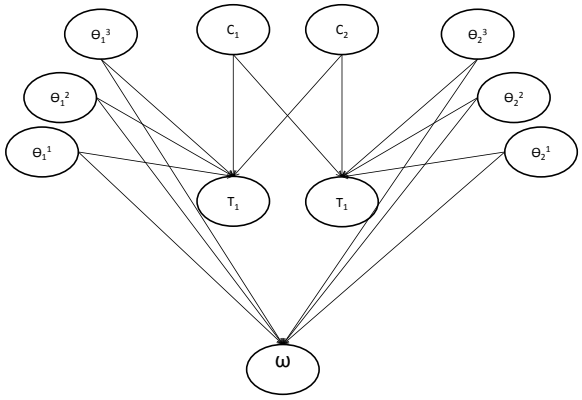


Figure 4.2: Augmented diagnostic Bayesian network with parameters and constraints

for the conditional distributions, the number of parameters are linear in the number of parents instead of exponential, as shown in Equation 4.2

$$\Pr(T_i = true|\mathbf{C}) = 1 - (1 - \theta_0^i) \prod_{j|C_j=true} (1 - \theta_j^i) \tag{4.2}$$

Here, $\theta_0^i = \Pr(T_i = true|C_j = false\ \forall j)$ is the *leak probability* that T_i will be true when none of the causes are true and $\theta_j^i = \Pr(T_i = true|C_j = true, C_k = false\ \forall k \neq j)$ is the *link reliability*, which indicates the independent contribution of cause C_j to the probability that test T_i will be true. In the rest of this chapter, it is described how to learn all the parameters, Θ, while respecting the constraints implied by test consistency.

The objective is to learn the Θ parameters of diagnostic Bayes networks given test constraints of the form described in Equations 2.4 and 4.1. To deal with non-convex constraints and disconnected feasible regions, a Bayesian approach is pursued whereby parameters and constraints are explicitly modeled as random variables in an augmented Bayes network (see Fig. 4.2).

In Bayesian learning, parameters are treated as random variables. The augmented Bayesian network thus contains a node corresponding to each parameter, θ, of the Bayesian network. The node corresponding to each θ is the parent of the variables for which it defines a conditional distribution. Now, θ can be estimated by computing the posterior over this node.

Similarly, the constraint is also modeled as a binary random variable, ω, such that its value is set to *True* if the constraint is satisfied and *False* otherwise. $Pr\left(\omega = true\right)$ defines a likelihood over the space of values for Θ. The objective then is to choose the values of Θ such that the value of $Pr(\omega = true|\Theta)$ is maximal in the interior of the feasible space and gradually decreases as we get closer to the boundaries. In one version, $Pr(\omega = true|\Theta)$ is set to 0 when Θ violates the constraints. In the other version it has a low probability which is useful when there is no feasible Θ.

The use of an augmented Bayesian network now allows framing the problem of learning the parameters as an inference problem in a hybrid Bayes network of discrete ($\mathbf{T}$, $\mathbf{C}$, ω) and continuous (Θ) variables. This augmented Bayesian network provides a unifying framework to simultaneously learn from constraints and data, to deal with possibly inconsistent constraints, and to express preferences over the degree of satisfaction of the constraints.

As mentioned earlier, if the variable ω is $True$ the constraint is satisfied; otherwise it is violated. Thus, if ω is *true* then Θ lies in the positive region of Fig. 4.5, and if ω is $false$ then $\mathbf{\Theta}$ lies in the negative region. This chapter presents two techniques to define the conditional probability table (CPT) for ω. In the first version, it is defined as $\Pr(\omega = True|\mathbf{\Theta}) = \max(0, \pi)$, where $\pi = \mathrm{GI}(\mathbf{C}|T_1) - \mathrm{GI}(\mathbf{C}|T_2)$. The term π provides information regarding the satisfiability of the constraint as per Equation 4.1 and it lies in the interval $[-1, 1]$ so $\max\left(\pi, 0\right)$ is always in $[0, 1]$, which can then be used to defined the probability of $\omega = True$. The intuition behind this definition of the CPT for ω is that a constraint is more likely to be satisfied if the parameters lie in the interior of the constraint region. Furthermore, if the constraint is violated, i.e., $\pi < 0$ then the probability is set to zero. The second version uses a more permissive approach in which the CPT for ω is defined as $Pr\left(\omega = True|\Theta\right) = \left(\pi + 1\right)/2$. This also ensures that the probability of ω is normalized. The first proposal explicitly prohibits any set of parameters that violate the constraint, no matter how small the violation is, whereas the latter caters for such violations and hence may be more suited for the case where an expert may make a mistake.

Note that each parameter, θ, is in fact a Bernoulli distribution. When computing its posterior using Gibbs sampling, it is helpful if the prior is expressed as its conjugate prior. Beta distribution is the conjugate prior of Bernoulli distribution. Thus, a Beta prior is placed over each θ parameter. Since the test variables are conditioned on the Θ parameters that are now part of the network, their conditional distributions become known. For instance, the conditional distribution for T_i (given in Equation 4.2) is fully defined given the noisy-or parameters θ^i_j. Hence the problem of learning the parameters becomes an inference problem to compute posteriors over the parameters given that the constraint is satisfied. In practice, it is more convenient to obtain a single value for the parameters instead of a posterior distribution since it is easier to make diagnostic predictions based on

one Bayes network. The parameters are estimated by computing a maximum a posteriori (MAP) hypothesis given that the constraint is satisfied: $\Theta^* = \arg\max_\Theta \Pr(\Theta|\omega = true)$.

The approach defined in this section maximizes the likelihood of satisfying the constraint given the parameters. Traditionally, parameter learning is performed by maximization of likelihood of data given the parameters. Gibbs sampling can be easily used to learn likelihood of parameters given data instead of constraints by simply discarding the nodes associated with constraints in the augmented Bayesian network (in Figure 4.2) and instead introducing nodes corresponding to the data samples, as shown in Figure 4.3. Each data sample will represent an observation that is an assignment of values to all the tests and causes. For the variables that have been observed the nodes will be restricted to their values and for variables that have not been observed their values will be sampled. Figure 4.3 depicts the case with one observed data point by introducing the additional nodes for each test and cause (T_i^d, C_i^d). If additional data points are available they can be added to the network in the same fashion by connecting them with the nodes for the parameters.

When both data and constraints are available it is also possible to maximize the likelihood of data and constraints simultaneously. The evaluation section will present results for all three approaches, i.e., learning with data only, learning with constraints only and learning with data and constraints.

The formulation of the problem based on the augmented network also allows for the case where an expert makes mistakes and the constraints are inconsistent. For the case where $Pr\left(\omega = True|\Theta\right) = \left(\pi + 1\right)/2$, the technique will still refine the model while maximizing the degree of satisfaction of all constraints simultaneously without any change. For the case where $\Pr(\omega = True|\boldsymbol{\Theta}) = \max(0, \pi)$, the rejection sampling step will reject all samples since no set of parameters will exist that satisfies all the constraints if the constraints are inconsistent. To account for the possibility of an expert making a mistake, a constraint can be considered as probabilistic such that probability of the expert being correct is represented as the confidence level of the expert. This can be accomplished by treating the observed feedback as a probabilistic indicator of the true constraint and thus adding an additional node in the Bayesian network to represent the true constraint that also needs to be sampled. This is shown in Figure 4.4 where ω^* represents the true constraint, ω is the observed constraint (possibly inconsistent with the true constraint), δ_{FP} is the probability of a false positive constraint and δ_{FN} is the probability of a false negative constraint.

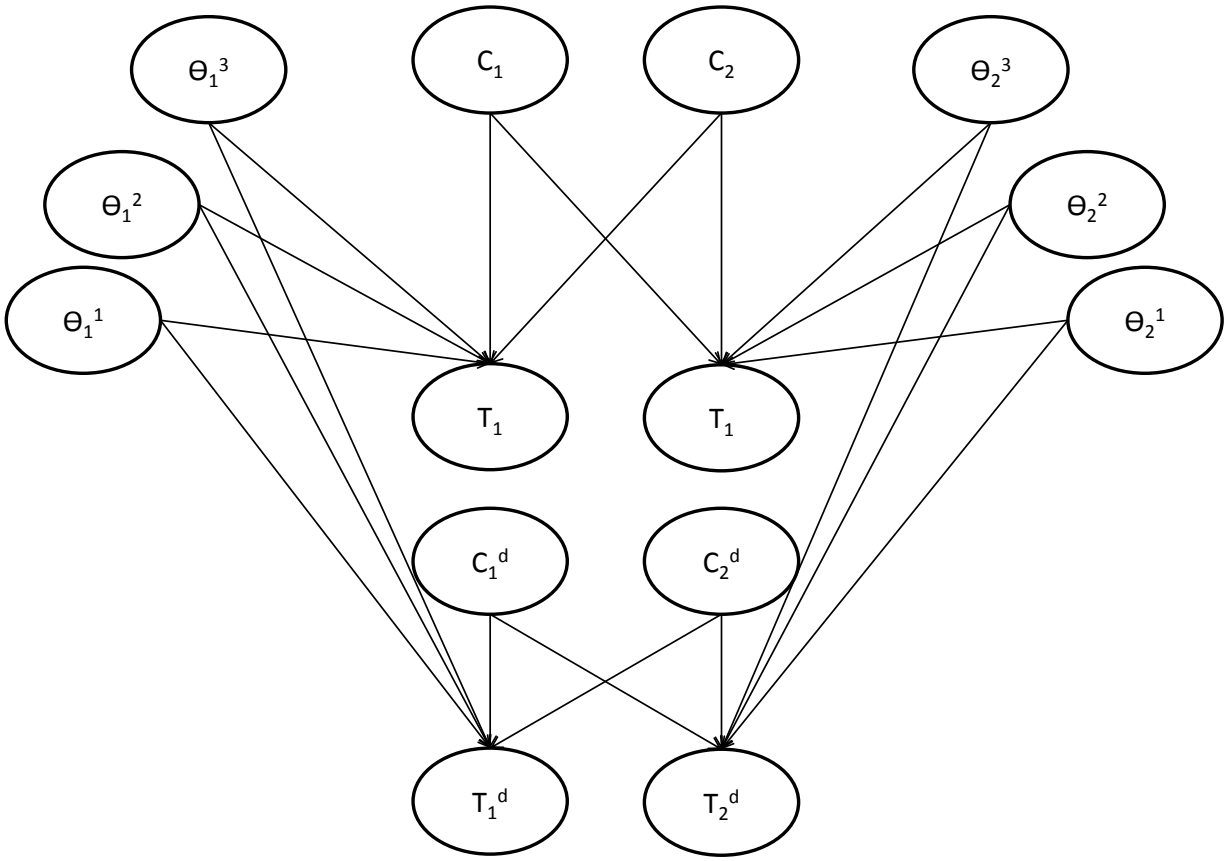
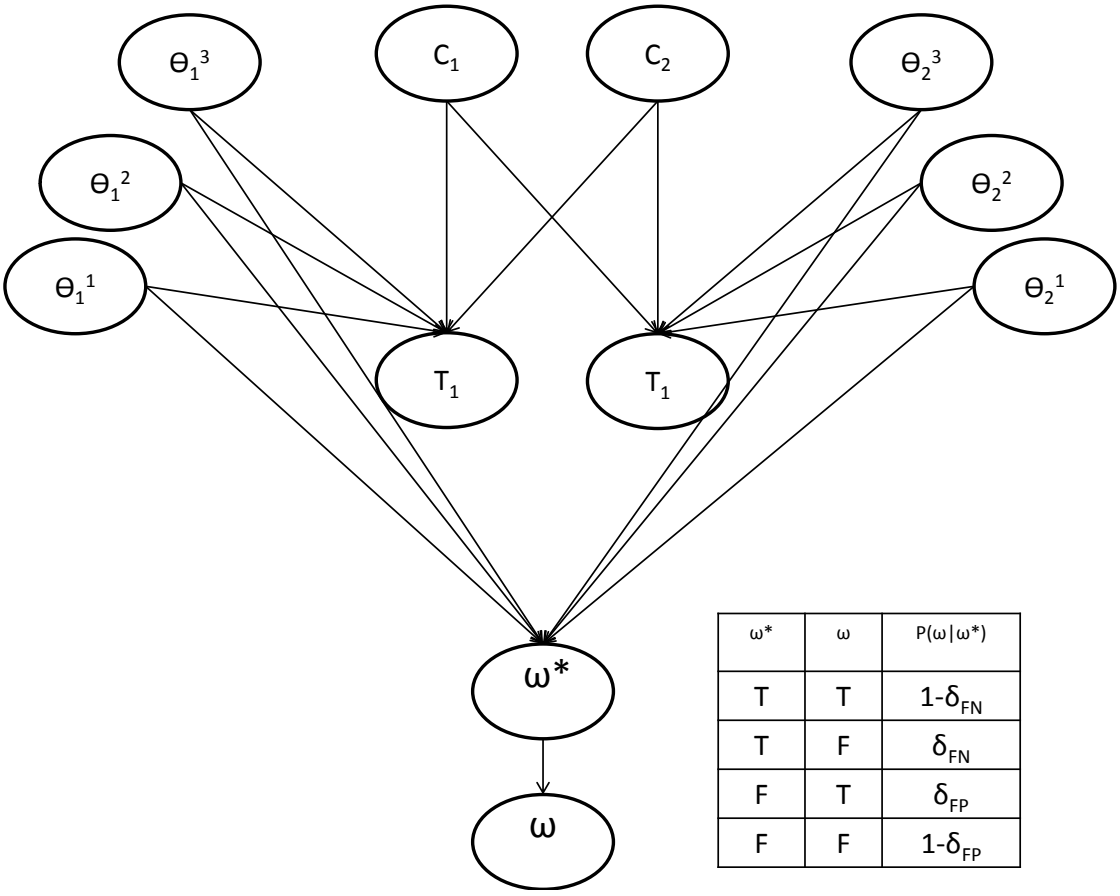


Figure 4.3: Augmented diagnostic Bayesian network with parameters and data



ω*	ω	P(ω|ω*)
T	T	$1-\delta_{FN}$
T	F	δ_{FN}
F	T	δ_{FP}
F	F	$1-\delta_{FP}$

Figure 4.4: Augmented diagnostic Bayesian network extended to handle inaccurate feedback

4.3.2 MAP Estimation

Previous approaches for parameter learning with domain knowledge include modified versions of EM or some other optimization techniques that account for linear/convex constraints on the parameters. Since constraints received from experts are non-convex (as shown in Figure 4.6), a new approach based on Gibbs sampling is proposed to approximate the posterior distribution, from which the MAP estimate is computed. Although the technique converges to the MAP in the limit, it may require excessive time when using rejection sampling. Hence, Gibbs sampling is modified to obtain more efficient stochastic hill climbing and greedy search algorithms with anytime properties.

Algorithm 2 Pseudo Code for Gibbs Sampling, Stochastic Hill Climbing and Greedy Search

```
1  Fix observed variables, let ω = true and randomly sample feasible starting state S
2  for i = 1 to #samples
3      for j = 1 to #hiddenVariables
4          acceptSample = false; k = 0
5          repeat
6              Sample s' from conditional of j^th hidden variable S_j
7              S' = S; S_j = s'
8              if S_j is cause or test, then acceptSample = true
9              elseif S' obeys constraints ω
10                 if algo == Gibbs
11                     Sample u from uniform distribution, U(0,1)
12                     if u < p(S')/(Mq(S')) where p and q are the true and proposal
                       distributions respectively and M > 1
13                         acceptSample = true
14                 elseif algo == StochasticHillClimbing
15                     if likelihood(S') > likelihood(S), then acceptSample = true
16                 elseif algo == Greedy, then acceptSample = true
17             elseif algo == Greedy
18                 k = k + 1
19                 if k == maxIterations, then s' = S_j; acceptSample = true
20         until acceptSample == true
21         S_j = s'
```

The pseudo code for this Gibbs sampler is provided in Algorithm 2. The two key steps are sampling the conditional distributions of each variable (line 6) and rejection sampling to ensure that the constraints are satisfied (lines 9 and 12). The rejection sampling is only required for the case where the conditional of ω has a max in it. Each variable is sampled given the rest according to the following distributions:

$$t_i \sim \Pr(T_i|\mathbf{c}, \theta_\mathbf{i}) \qquad \forall i \tag{4.3}$$

$$c_j \sim \Pr(C_j|\mathbf{c} - c_j, \mathbf{t}, \theta) \propto \prod_j \Pr(C_j) \prod_i \Pr(t_i|\mathbf{c}, \theta_\mathbf{i}) \qquad \forall j \tag{4.4}$$

$$\theta^i_j \sim \Pr(\Theta^i_j|\mathbf{\Theta} - \Theta^i_j, \mathbf{t}, \mathbf{c}, \omega) \propto \Pr(\omega|\mathbf{t}, \mathbf{\Theta}) \prod_i \Pr(t_i|\mathbf{c}_j, \theta_i) \qquad \forall i, j \tag{4.5}$$

The tests and causes are easily sampled from the multinomials as described in the equations above. However, sampling the θ's is more difficult due to the factor $\Pr(\omega|\Theta, \mathbf{t}) = \max(0, \pi)$, which is a truncated mixture of Betas. So, instead of sampling θ from its true conditional, it is sampled from a proposal distribution that replaces $\max(0, \pi)$ by an untruncated mixture of Betas equal to $\pi + a$ where a is a constant that ensures that $\pi + a$ is always positive. This is equivalent to ignoring the constraints. Then to ensure that the constraints are satisfied, samples that violate the constraints are rejected. Rejection sampling is not required when $Pr\left(\omega|\Theta, \mathbf{t}\right) = \left(\pi + 1\right)/2$ and thus the check in line 9 can be skipped.

Once Gibbs sampling has been performed, a sample that approximates the posterior distribution over the parameters given the constraints (and any data) is obtained. A single setting of the parameters is returned by selecting the sampled instance with the highest posterior probability (i.e., MAP estimate). Since this will only return the MAP estimate, it is possible to speed up the search by modifying Gibbs sampling. In particular, a stochastic hill climbing algorithm can be used by accepting a new sample only if its posterior probability improves upon that of the previous sample (line 15). Thus, each iteration of the stochastic hill climber requires more time, but always improves the solution.

As the number of constraints grows and the feasibility region shrinks, the Gibbs sampler and stochastic hill climber with rejection sampling will reject most samples. This can be mitigated by using a Greedy sampler that caps the number of rejected samples, after which it abandons the sampling for the current variable to move on to the next variable (line 19). Even though the feasibility region is small overall, it may still be large in some dimensions, so it makes sense to try sampling another variable (that may have a larger range of feasible values) when it is taking too long to find a new feasible value for the current variable.

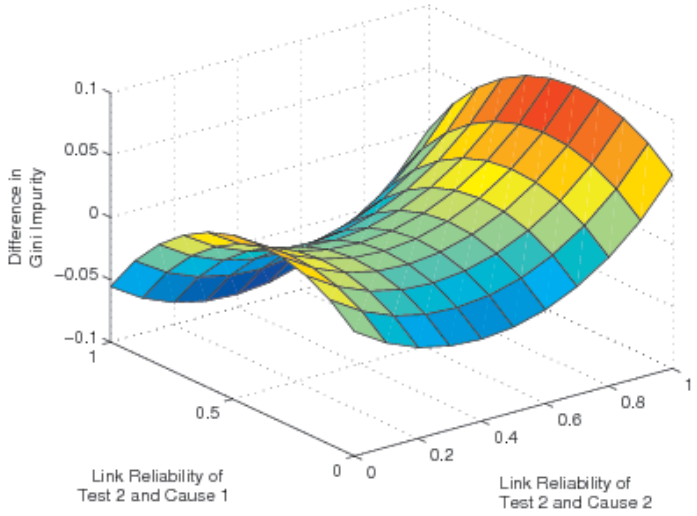


Figure 4.5: Difference in Gini impurity for the network in Fig. 4.1 when θ_2^1 and θ_2^2 are the only parameters allowed to vary.

4.4 Evaluation and Experiments

Formally, for M^*, the true model that needs to be learnt, the diagnostic process determines the choice of best next test as the one with the smallest Gini impurity. If the correct choice for the next test is known (such as demonstrated by an expert), this information can be used impose a constraint on the model. Let $\mathbf{\Gamma}^+$ denote the set of observed constraints and $\mathbf{\Gamma}^*$ the set of all possible constraints that hold for M^*. Having only observed $\mathbf{\Gamma}^+$, the technique will consider any $M^+ \in \mathbf{M}^+$ as a possible true model, where $\mathbf{M}^+$ is the set of all models that obey Γ^+. Let $\mathbf{M}^*$ denote the set of all models that are *diagnostically equivalent* to M^* (i.e., obey Γ^* and would recommend the same steps as M^*) and by $M^{\text{MAP}}_{\mathbf{\Gamma}^+}$ the particular model obtained by MAP estimation based on the constraints $\mathbf{\Gamma}^+$. Similarly, when a dataset $\mathcal{D}$ is available, $M^{\text{MAP}}_{\mathbf{D}}$ denotes the model obtained by MAP estimation based on $\mathbf{D}$ and $M^{\text{MAP}}_{\mathbf{D}\mathbf{\Gamma}^+}$ the model based on $\mathbf{D}$ and $\mathbf{\Gamma}^+$.

The objective is to learn the model M^*, hence the KL divergence between the models found and M^* is reported. However, other *diagnostically equivalent* M^* may recommend the same tests as M^* and thus have similar constraints, so *test consistency* with M^* (i.e., # of recommended tests that are the same) is also reported.

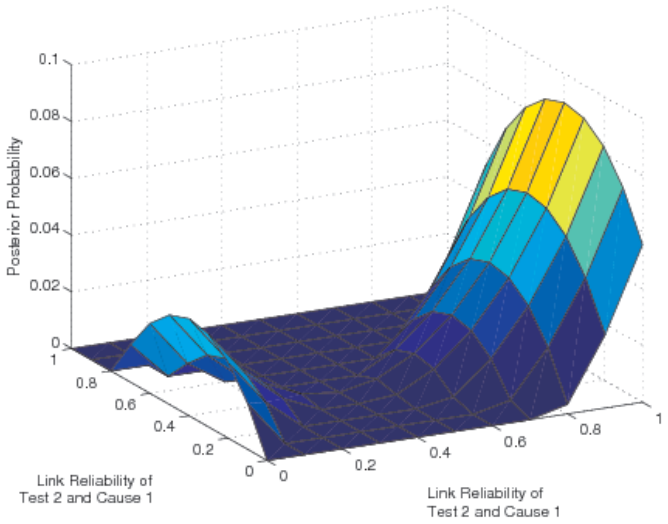


Figure 4.6: Posterior of parameters of network in Figure 4.2 calculated through discretization.

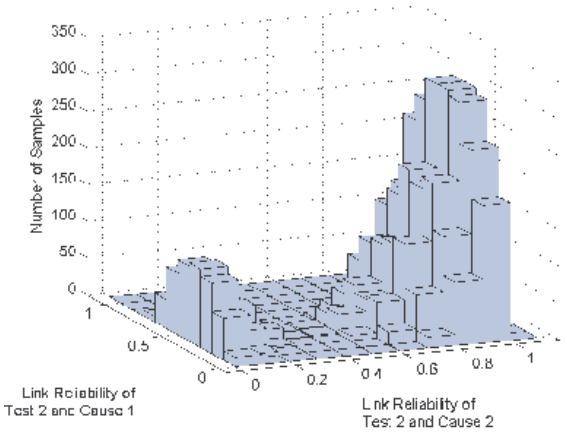


Figure 4.7: Posterior of parameters of network in Figure 4.2 estimated through sampling.

4.4.1 Correctness of Model Refinement

Given $\mathbf{\Gamma}^*$, the technique for model refinement with $Pr(\omega|\Theta) = \max(0, \pi)$ is guaranteed to choose a model $M^{\text{MAP}} \in \mathbf{M}^*$ by construction. If any constraint $\Gamma^* \in \mathbf{\Gamma}^*$ is violated, the rejection sampling step of the technique would reject that set of parameters. To illustrate this, consider the network in Figure 4.2. There are six parameters (four link reliabilities and two leak parameters). If the leak parameters and the link reliability from the first cause to each test is fixed, then the posterior surface over the two variable parameters can be computed after discretizing each parameter in small steps and then calculating the posterior probability at each step as shown in Figure 4.6. This surface can be compared with that obtained after Gibbs sampling using the refinement technique as shown in Figure 4.7. It can be seen that the refinement technique recovers the posterior surface from which the MAP is then computed.

4.4.2 Experimental Results on Synthetic Problems

This section presents results on a 3-cause by 3-test fully-connected bipartite Bayes network. It is assumed that there exists some $M^* \in \mathbf{M}^*$ that needs to be learnt given $\mathbf{\Gamma}^+$. The refinement process described in this chapter is then used find M^{MAP}. First constraints, $\mathbf{\Gamma}^*$ for M^* are computed to get the feasible region associated with the true model. Next, 100 other models are sampled from this feasible region that are diagnostically equivalent. These models are then compared with M^{MAP} (after collecting 200 samples with non-informative priors for the parameters using the refinement technique).

Figure 4.8 compares the convergence rate of each technique to find the MAP estimate. Gibbs sampling based on the technique without rejection sampling ($Pr(\omega|\Theta, \mathbf{t}) = (\pi + 1)/2$) is quite fast since there is no rejection sampling. Gibbs sampling with rejection sampling ($Pr(\omega|\Theta, \mathbf{t}) = \max(\pi, 0)$) is slower due to rejection sampling. However, the use of stochastic hill climbing and greedy sampling can speed up the process considerably. Also, note that the log-likelihood shown for the case without rejection sampling converges to a different value than the other three techniques with rejection sampling since the conditional for ω is defined differently. In Figure 4.8 the log likelihood for the case without rejection sampling is normalized on the same scale as the other approach so the convergence results can be compared against each other. Also note that while the values for the log-likelihood are different for the two variants of the CPT of ω, the same set of parameters will be returned in all cases when the techniques converge.

The KL-divergence of M^{MAP} with respect to each sampled model is computed. The KL-divergence should decrease as the number of constraints in $\mathbf{\Gamma}^+$ increases since the feasible

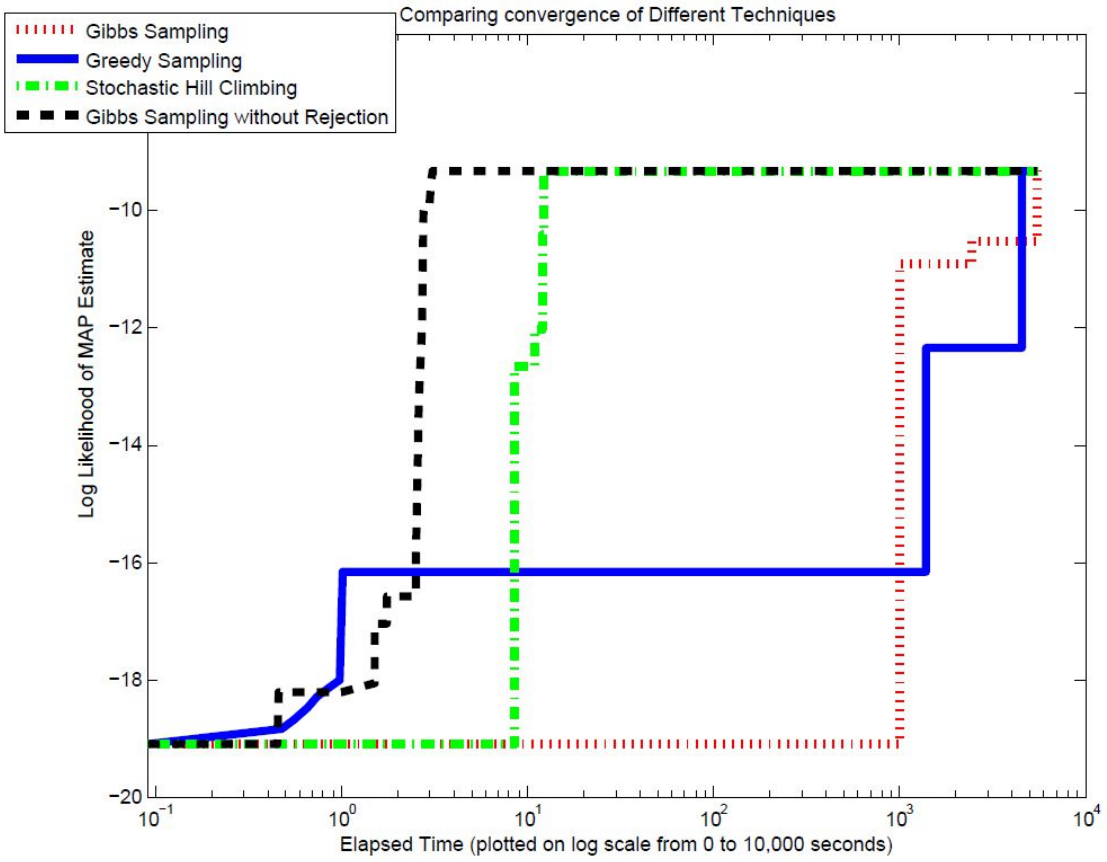


Figure 4.8: Comparison of Convergence Rates for Model Refinement Techniques for Bayesian Networks

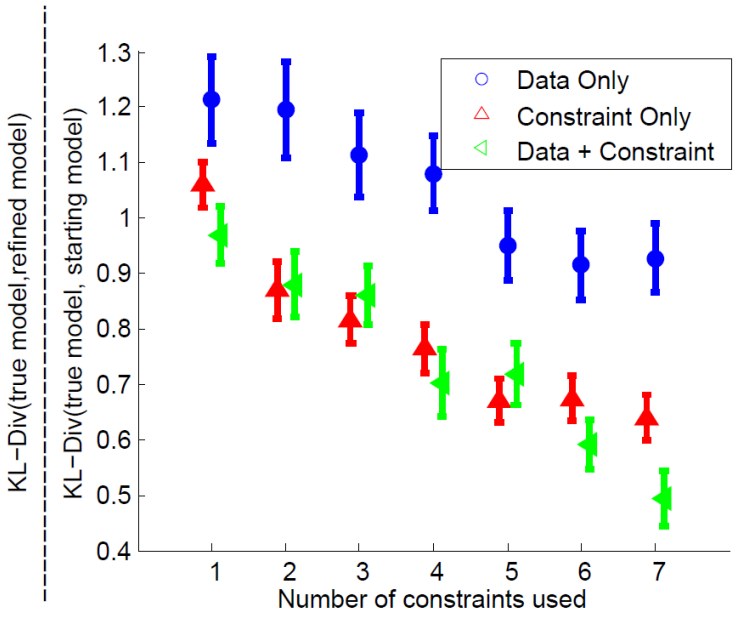


Figure 4.9: Ratio of KL-divergence of True Model with Refined Model after Model Refinement for Bayesian Networks to True Model with Initial Model – Synthetic Diagnostic Problem

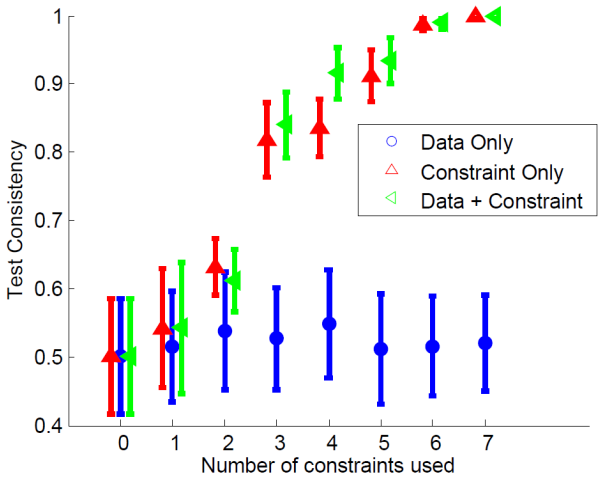


Figure 4.10: Test Consistency after Model Refinement for Bayesian Networks – Synthetic Diagnostic Problem

region becomes smaller. Figure 4.9 confirms this trend and shows that $M^{\text{MAP}}_{\mathbf{D}\boldsymbol{\Gamma}^+}$ has lower mean KL-divergence than $M^{\text{MAP}}_{\boldsymbol{\Gamma}^+}$, which has lower mean KL-divergence than $M^{\text{MAP}}_{\mathbf{D}}$. The data points in $\mathbf{D}$ are limited to the results of the diagnostic sessions needed to obtain $\boldsymbol{\Gamma}^+$. As constraints increase, more data is also available and so the results for the approach that combines data and constraints improves. The results shown in Figure 4.9 are for the Gibbs sampling approach without any rejection sampling (results are similar for the other techniques since they return the same MAP estimate).

The test consistency is also compared for the cases when learning from data only, constraints only or both. Given a fixed number of constraints, the unobserved trajectories are enumerated, and then the highest ranked test is computed using the learnt and sampled true models, for each trajectory. The test consistency is reported as a percentage, with 100% consistency indicating that the learned and true models had the same highest ranked tests on every trajectory. Figure 4.10 presents these percentages for the Gibbs sampling approach without rejection (the results are again similar for the other techniques since they return the same MAP estimate). It again appears that learning parameters with both constraints and data is better than learning with only constraints, which is most of the times better than learning with only data. Note that as more constraints are available, the possibility of test consistency decreases when using constraints as only a few unobserved trajectories remain over which the test consistency is computed. This is why when all constraints are known, the test consistency is 100% with no possibility of any error.

Gibbs sampling without rejection converges to the same posterior as the other approaches, but before convergence, it is possible that the best MAP estimate available may not satisfy all the constraints. This is because this approach does not explicitly rule out all parameter settings which violate any constraint. All approaches with rejection sampling ensure that the best estimate for the MAP at any time does not violate any constraints from the expert. Any set of parameters that do violate the constraint would be rejected in the rejection step.

4.4.3 Experimental Results on Large Scale Diagnostic Problems

The refinement technique presented in this chapter is also evaluated on a real-world diagnostic network collected and reported by Agosta et al. [6], where the authors collected detailed session logs over a period of seven weeks in which the entire diagnostic sequence was recorded. The sequences intermingle model building and querying phases. The model network structure was inferred from an expert's sequence of positing causes and tests. Test-ranking constraints were deduced from the expert's test query sequences once the network structure is established.

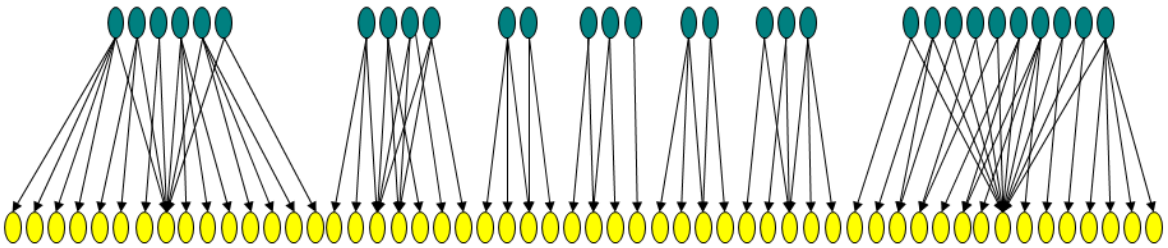


Figure 4.11: Large Scale Diagnostic Bayesian Network

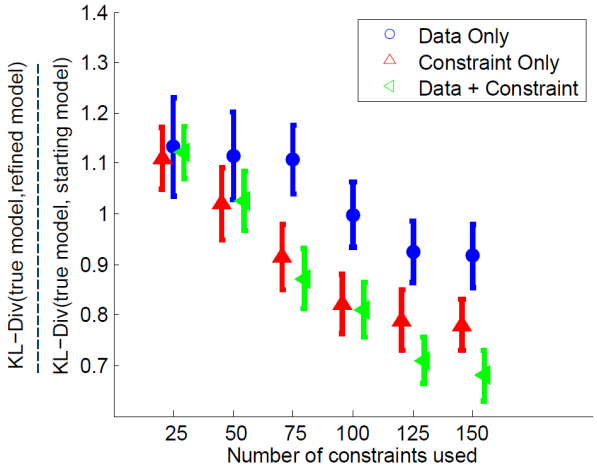


Figure 4.12: Ratio of KL-divergence of True Model with Refined Model after Model Refinement for Bayesian Networks to True Model with Initial Model – Large Scale Diagnostic Problem

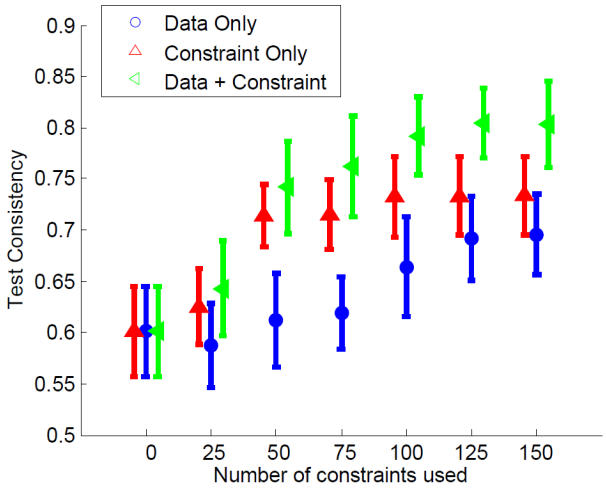


Figure 4.13: Test Consistency after Model Refinement for Bayesian Networks – Large Scale Diagnostic Problem

The 157 sessions captured over the seven weeks resulted in a Bayes network with 115 tests, 82 root causes and 188 arcs. The network consists of several disconnected sub-networks, each identified with a symptom represented by the first test in the sequence, and all subsequent tests applied within the same subnet. There were 20 sessions from which it was possibled to observe trajectories with at least two tests, resulting in a total of 32 test constraints. The diagnostic network was pruned to remove the sub-networks with no constraints to get a Bayes network with 54 tests, 30 causes, and 67 parameters divided in 7 sub-networks, as shown in Figure 4.11, on which the model refinement technique is applied to learn the parameters for each sub-network separately.

Since the true underlying network is not available, the full set of 32 constraints were treated as if they were $\mathbf{\Gamma}^*$ and the corresponding feasible region $\mathbf{M}^*$ as if it contained models diagnostically equivalent to the unknown true model. Figure 4.12 reports the KL divergence between the models found by the algorithms and sampled models from $\mathbf{M}^*$ as the number of constraints is increased. It can be seen that the ratio decreases slightly for the case where only data is used as more data is collected. However, using only constraints provides better performance. Furthermore, as the constraints increase (and the data used also increases), the approach that combines the use of data and constraints, provides better results than using constraints alone. Similar effects are also observed in Figure 4.13 for test consistency where using constraints provides better results than using data alone. Again, as more constraints are available (and more data is also consequently available),

the results for the approach that combines data with constraints are better than using only constraints or data. Unlike Figure 4.10, the test consistency does not reach 100% because in this scenario, not all possible constraints have been incorporated (constraints are exponential in the state space and cannot be enumerated for large problems).

4.5 Summary

The problem of learning by example has the promise to create strong models from a restricted number of cases; certainly humans show the ability to generalize from limited experience. Such approaches are especially important for domains in which the data has to be collected through interactions with users. Not much data is likely to be available, and even more so in domains such as diagnosis where faults are a rarity in any case. This chapter presents an approach to learn the parameters of Bayesian networks by observing the actions of experts in similar cases. These observations from experts are then modeled as constraints on the parameters of the model.

Several approaches have been proposed to incorporate qualitative constraints on parameters for Bayesian networks. However, the constraints derived from observing experts are non-convex in nature and thus require a different approach. This chapter presents different techniques to incorporate these constraints implied by test ordering available from experts in addition to using any data that may be available. The three major contributions of this work are as follows. First, this is the first approach that exploits implicit constraints based on value of information assessments. Secondly it is the first approach that can handle non-convex constraints in learning the parameters of Bayesian networks. Third, the approach is demonstrated on synthetic data as well as a real-world manufacturing diagnostic problem. Since data is generally sparse in such domains, this work makes an important advance to mitigate the model acquisition bottleneck, which has prevented the widespread application of diagnostic networks so far.

The technique presented in this chapter is explained in the context of a diagnostic Bayesian network. Diagnostic Bayesian networks are typically parameterized using Noisy-OR CPTs. The refinement technique presented in this chapter can be as easily applied to a Bayesian network that uses a complete parameterization by representing each parameter explicitly in the augmented Bayesian network. Similarly, the technique is also not specific to diagnostic Bayesian networks. It can be applied to a Bayesian network for any other recommender system by representing the parameters for that network with out any loss of generality. The only implementation detail that will require a change is the sampling of variables (Equations 4.3, 4.4 and 4.5) depending on the structure of the Bayesian network.

As mentioned earlier, Bayesian networks are not useful for sequential problems. The next chapter addresses model refinement for sequential problems using MDPs and presents an approach to incorporate these constraints from expert feedback in MDPs.

Chapter 5

Refining Models for Markov Decision Processes

Consider the problem of diagnostics where a Markov decision process is available. This MDP can be used to compute a policy that will select the optimal action based on a reward function. The reward function may optimize for disambiguating the cause as soon as possible, executing tests with minimal costs, being risk averse to rule out the more critical faults first, or some combination of these and any other factors. If the decision-making is being performed by an expert, the expert would also be trying to optimize for a similar objective. So if an expert can be observed performing some tests, with the assumption that the tests are being selected optimally, then the MDP model should also be consistent with the expert. The consistency can be evaluated by computing the MDP policy with the highest value and comparing it against the expert's choice. This chapter examines this problem.

A preliminary version of parts of this chapter have been accepted for publication in the proceedings of 2013 European Conference on Machine Learning and Principles of Knowledge Discovery in Databases (ECML-PKDD) [50].

5.1 Problem Statement

MDPs are designed by defining the state variables, the actions, and then estimating/specifying transition and reward functions, as well as a discount factor. In this chapter, the reward function ρ and the discount factor γ are assumed to be specified accurately, while the

transition function is imprecise. The imprecise transition function is denoted as $\tilde{T}$ and the resulting imprecise MDP is denoted as $\tilde{M} = \langle S, A, \tilde{T}, \rho, \gamma \rangle$. Let the true underlying MDP be denoted as $M = \langle S, A, T, \rho, \gamma \rangle$, where T is the actual transition function. Since $\tilde{T}$ is imprecise, the optimal policy for $\tilde{M}$, $\tilde{\pi}^*$, may also not be optimal for M. As the expert reviews the policy for $\tilde{M}$, she can point out non-optimal actions and specify true optimal actions for those states, which would reflect π^*. These observations from experts can be treated as constraints, where each constraint is represented as a state-action pair, $\langle s, a \rangle$, which indicates the true optimal action for that state. The objective in refining the transition function is to modify $\tilde{T}$ to $\hat{T}$ such that the optimal policy $\hat{\pi}^*$ for this new MDP $\hat{M} = \langle S, A, \hat{T}, \rho, \gamma \rangle$ obeys all constraints, and thus matches the true optimal policy for these states, *i.e.*, $\hat{\pi}^*(s) = \pi^*(s)$.

5.2 Literature Review

The idea of learning and refining an MDP model or a policy based on expert feedback or demonstration has been widely used, but the focus has mostly been to either learn the reward function or otherwise learn the optimal policy without learning the reward function. This section reviews some of these techniques to learn or refine the model.

5.2.1 Inverse Reinforcement Learning and Apprenticeship Learning

Inverse reinforcement learning(IRL) deals with recovering a reward function using a known policy and transition function [84, 68]. The assumption here is that the optimal policy is implicitly using a fixed reward function. Inverse reinforcement learning can then be framed as a linear optimization problem where the constraint is $Q^{\pi}(s, \pi^*(s)) \geq Q^{\pi}(s, a') \quad \forall a' \neq \pi^*(s)$. In the absence of a meaningful objective function, Ng and Russell [68] show that a trivial solution to this problem is setting the reward function to zero as that would satisfy any policy. To overcome this issue, meaningful penalty terms and regularization need to be used, such as penalizing single-step deviations from the optimal policy by making them as costly as possible or rewarding smaller reward values to higher ones.

IRL assumes the existence of a known optimal policy which is often not possible. The other major difference is the assumption of a known transition function which is then used to learn an unknown reward function. Learning the reward function can be framed as a linear optimization problem whereas learning the transition function with a known reward

function corresponds to a non-convex optimization problem, as will be discussed later in this chapter.

Apprenticeship learning [3] is the process of observing an expert demonstrate a policy, usually only partially, to learn the policy that would be obtained using the reward function that the expert is trying to optimize. This is done without explicitly learning the reward function that the expert is supposedly using. The approach here is to represent the reward function as a linear combination of some pre-defined features and then to learn the weights of these features by finding a maximum margin hyperplane separating two data points as in SVMs [96].

The problem of learning a reward function with a known transition function and observations from experts has also been posed as a maximum margin planning problem such that the margin between the value of the expert's policy and other alternate policies is increased [81]. This approach attempts to learn a direct mapping between the features and the reward functions while minimizing a loss function.

5.2.2 Reinforcement Learning with Expert Feedback

Abbeel and Ng [4] present a technique to learn the dynamics of a system after observing multiple expert trajectories. Their technique involves running several trials using the expert's policy and then saving all the produced trajectories. A maximum likelihood technique is used on these state-action trajectories to estimate the transition function. The estimated transition function is then used to compute a policy which is compared with the expert's policy. If the policies are not close enough the process is repeated by re-estimating the transition function through maximum likelihood.

Imitation learning [77, 78] is a similar concept that has been used to accelerate the process of reinforcement learning by differentiating between a mentor and observer. The observer is assumed to have a known reward function but an unknown transition function that it is trying to learn by observing the mentor. The observer updates its belief about a state transition after observing the transition from a mentor and also can update its value function using an augmented Bellman backup. The augmented backup contains a term to indicate the expected value of duplicating the mentor's action.

Knox and Stone [54, 55] present a technique that explicitly incorporates human generated feedback on the policy. This feedback is received in the form of a critique of a state-action pair generated as a result of the policy, where the critique can either be positive or negative. Additionally, this critique is focused on reward functions and is used to learn the optimal policy rather than refine the transition function.

Approaches based on Bayesian reinforcement learning have also been proposed [11, 27] that incorporate feedback from experts. A set of models are initially sampled from the prior distribution and then a Bayesian update is performed to compute a posterior over the distribution of the parameters for each model. The most unlikely models are then discarded in favor of freshly sampled models.

Reinforcement learning combined with expert constraints is similar to the problem being addressed in this chapter, except these approaches assume the availability of significant feedback from experts. They are well-suited for problems of control in robotics and other domains where a lot of data can be collected using limited interaction with an expert (such as receiving hundreds of readings per second while observing a user drive a car) but not for cases where this feedback is limited (such as diagnostics), which is likely the case for recommender systems.

5.2.3 Imprecise and Robust MDPs

There is a wide section of literature that focuses on solving MDPs with imprecise probabilities (MDP-IP) [85, 99, 95, 24] where transition probabilities are modeled as distributions. Combined with some optimization criterion such as max-min, these distributions are then used to produce a robust solution. Such solutions are generally computationally intensive and do not scale well. Bounded parameter MDPs [39, 93] are an extension of MDP-IPs such that the upper and lower bounds on the parameters are provided on the probabilities of the transition parameters (as well as the reward functions).

A related section of literature focuses on accepting that the model is imprecise. Abbeel et al., [5] present an approach that makes local improvements to the current policy using techniques such as policy gradient methods. Nilim and Ghaoui [70] allow explicit representation of the uncertainty on the transition matrices and present a technique to optimize robust policies for MDPs under this uncertainty. Xu and Mannor [102] frame the problem of robust control as a trade-off between nominal versus worst case performance. This ensures the robust policies are not unnecessarily conservative. Delage and Mannor [23] formalize this trade-off and present an approach under which policies for MDPs can be computed using percentile optimization. This provides a formal guarantee about the performance of the policy. Other approaches based on the preference elicitation framework have also been proposed to compute policies that are robust to the uncertainty in the reward function of an MDP [82].

5.2.4 Constrained MDPs

There has been some work on representing MDPs with certain constraints [9, 37, 75, 20, 42]. These constraints are typically linear and allow multi-objective criteria to be satisfied such as ensuring a maximum loss or minimum gain on certain attributes while maximizing the utility otherwise. Techniques based on dynamic programming or linear optimization are generally used to solve these MDPs. The constraints explored in this thesis based on expert feedback are non-convex and cannot be satisfied using dynamic programming or linear optimization.

5.3 Model Refinement

Let Γ be the set of constraints obtained from expert feedback of the form $\langle s, a^* \rangle$, which means that executing a^* should have a value at least as high as any other action in s.

$$Q^{\hat{\pi}^*}(s, a^*) \geq Q^{\hat{\pi}^*}(s, a) \quad \forall a \tag{5.1}$$

Section 5.3.1 explains the technique to refine the transition model based on such constraints for flat MDPs and Section 5.3.2 for factored MDPs .

5.3.1 Flat Model Refinement

An optimization problem can be set up to find a refined transition model $\hat{T}$ that maximizes the gap δ between optimal and non-optimal Q-values as specified by the expert's constraints.

$$\max_{\hat{T}, \delta} \ \delta \quad \text{s.t.} \quad Q^{\hat{\pi}}(s, a^*) \geq Q^{\hat{\pi}}(s, a) + \delta \quad \forall \langle s, a^* \rangle \in \Gamma, \forall a \tag{5.2}$$

When δ is non-negative, the refined model satisfies the expert's constraints. If the user's constraints are inconsistent, this optimization will simply find a model that minimizes the maximum degree of violation for any constraint. For problems with a finite horizon h, the Q function can be re-written as a sum of expected rewards

$$Q^{\pi}(s_0, a_0) = \rho(s_0, a_0) + \sum_{t=1}^{h} \gamma^t \sum_{s_t} \Pr(s_t | s_0, a_0, \pi) \rho(s_t, \pi(s_t)) \tag{5.3}$$

where the probability $\Pr(s_t|s_0, a_0)$ is obtained by a product of transition probabilities.

$$\Pr(s_t|s_0, a_0, \pi) = \sum_{s_{1..t-1}} \Pr(s_1|s_0, a_0) \prod_{i=2}^{t} \Pr(s_i|s_{i-1}, \pi(s_{i-1})) \tag{5.4}$$

In a flat MDP, the transition probabilities are the transition parameters. Hence, θ can be used to denote the vector of transition parameters $\theta_{s'|s,a} = \Pr(s'|s, a)$. The optimization problem (5.2) can then be re-written in terms of θ by substituting Equations 5.3 and 5.4:

$$\max_{\hat{\theta},\delta} \delta$$

$$\text{s.t.} \quad \sum_{s'} \hat{\theta}_{s'|s,a} = 1 \quad \forall s, a \tag{5.5}$$

$$\hat{\theta}_{s'|s,a} \geq 0 \quad \forall s, a, s' \tag{5.6}$$

$$\rho(s, a^*) + \sum_{t=1}^{h} \gamma^t \sum_{s_{1..t}} \Pr(s_1|s, a^*) \prod_{i=2}^{t} \theta_{s_i|s_{i-1},\pi(s_{i-1})} \rho(s_t, a_t) \geq$$

$$\rho(s, a) + \sum_{t=1}^{h} \gamma^t \sum_{s_{1..t}} \Pr(s_1|s, a) \prod_{i=2}^{t} \theta_{s_i|s_{i-1},\pi(s_{i-1})} \rho(s, a) + \delta \quad \forall \langle s, a^* \rangle \in \Gamma, \forall a \tag{5.7}$$

This optimization problem is non-linear (and in fact non-convex) due to the product of θ's in the last constraint (equation 5.7.)

This section presents an approach to tackle the problem by alternating optimization where a subset of the parameters are iteratively optimized while keeping the remaining parameters fixed. This approach takes advantage of the fact that states in recommender systems can be organized in levels to do this. As explained earlier in Section 2.5.2, states are never visited twice since at each step one more variable is observed. Since each transition parameter $\theta_{s'|s,a}$ is associated with a state s, the transition parameters can also be partitioned into levels and the same transition parameter won't occur more than once in any state trajectory. Hence, if only the parameters in level l are varied while keeping the other parameters fixed, the Q function of the state-action pair of any constraint before level l can be written as a linear function of the θ's in level l.

$$Q(s, a^*) = c(nil) + \sum_{s_l, a_l, s_{l+1}} c(s_l, a_l, s_{l+1}) \theta_{s_{l+1}|s_l,a_l}$$

Here, $c(s_l, a_l, s_{l+1})$ is the coefficient of $\theta_{s_{l+1}|s_l,a_l}$ and $c(nil)$ is a constant. Algorithm 3 describes how to compute the coefficients of the parameters at level l for the Q function at

level $j \leq l$. First, the value function at level $l+1$ is computed by value iteration, then the coefficients for the Q function at level l are initialized and finally the coefficients of the Q functions at previous levels are computed by dynamic programming.

Algorithm 3 Linear dependence of the Q function at level j on the θ's at level l

LEVELLINEARDEPENDENCE(j, l, π)

Compute $V^\pi(s_{l+1})\ \forall s_{l+1}$
$V^\pi(s_h) = \rho(s_h, \pi(s_h))\ \forall s_h$
for $t = h-1$ down to $l+1$
    $V^\pi(s_t) \leftarrow \rho(s_t, \pi(s_t)) + \gamma \sum_{s_{t+1}} \theta_{s_{t+1}|s_t,\pi(s_t)} V^\pi(s_{t+1})\ \forall s_t$
Initialize the coefficients for the Q function at level l
for each s_l, a_l
    $c_{s_l,a_l}(nil) \leftarrow \rho(s_l, a_l)$
    $c_{s_l,a_l}(s_l, a_l, s_{l+1}) \leftarrow \gamma V(s_{l+1})\ \forall s_{l+1}$
    $c_{s_l,a_l}(s, a, s') \leftarrow 0\ \forall \langle s, a\rangle \neq \langle s_l, a_l\rangle, \forall s'$
Compute the coefficients for the Q function at levels before l
for $t = l-1$ down to j
    for each s_t, a_t
        $c_{s_t,a_t}(nil) \leftarrow \rho(s_t, \pi(s_t)) + \gamma \sum_{s_{t+1}} \theta_{s_{t+1}|s_t,\pi(s_t)} c_{s_{t+1},\pi(s_{t+1})}(nil)$
        $c_{s_t,a_t}(s_l, a_l, s_{l+1}) \leftarrow \gamma \sum_{s_{t+1}} \theta_{s_{t+1}|s_t,\pi(s_t)} c_{s_{t+1},\pi(s_{t+1})}(s_l, a_l, s_{l+1})\ \forall s_l, a_l, s_{l+1}$
return c

If the optimization problem (5.5) is restricted to the parameters at level l, the result is a linear program (5.8) since the last constraint expresses an inequality between pairs of Q functions that are linear combinations of the coefficients at level l.

$$\max_{\hat{\theta},\delta} \delta \quad \text{s.t.} \quad \sum_{s'} \hat{\theta}_{s_{l+1}|s_l,a_l} = 1\ \forall s, a \qquad \hat{\theta}_{s_{l+1}|s_l,a_l} \geq 0\ \forall s_l, a_l, s_{l+1} \tag{5.8}$$

$$c_{s,a^*}(nil) + \sum_{s_l,a_l,s_{l+1}} c_{s,a^*}(s_l, a_l, s_{l+1}) \hat{\theta}_{s_{l+1}|s_l,a_l} \geq$$

$$c_{s,a}(nil) + \sum_{s_l,a_l,s_{l+1}} c_{s,a}(s_l, a_l, s_{l+1}) \hat{\theta}_{s_{l+1}|s_l,a_l} + \delta \quad \forall \langle s, a^*\rangle \in \Gamma$$

To summarize, instead of directly solving the non-linear optimization problem (5.5), an alternating optimization technique (Algorithm 4) is used that solves a sequence of linear

programs (5.8) by varying only the parameters at one level. The algorithm continues until the gap δ is non-negative or until convergence. There is no guarantee that a feasible solution will be found, but each iteration ensures that δ will increase or remain constant, meaning that the degree of inconsistency is monotonically reduced. Given the non-convex nature of the optimization, random restarts are employed to increase the chances of finding a model that is as consistent as possible with the expert's constraints.

Algorithm 4 Alternating optimization to reduce the degree of inconsistency of the transition model with the expert's constraints in flat MDPs

ALTERNATINGOPT

repeat
  Initialize θ randomly
  repeat
    for $l = 1$ to h
      Compute coefficients for level l according to Algorithm 3
      $\delta, \{\theta_{s_{l+1}|a_l,s_l}\} \leftarrow$ solve LP (5.8) for level l
  until convergence
until $\delta \geq 0$
return θ

5.3.2 Factored Model Refinement

The approach described in the previous section assumes that the Markov decision process is flattened. This will only scale for small problems with a few recommendations and observations since the number of states grows exponentially with the number of choices. This sections presents a variant for problems with a large number of recommendations that avoids flattening by working directly with a factored model. The transition function is assumed to be factored into a product of conditional distributions for each variable X'_i given its parents $par(X'_i)$.

$$Pr(s'|s,a) = \prod_i Pr(X'_i|par(X'_i))$$

Furthermore, the parents of each variable are assumed to be a small subset of all the variables. For instance, in a course advising domain, the grade of a course may depend

only on the grades of the pre-requisites. As a result, the total number of parameters for the transition function shall be polynomial in the number of variables even though the number of states is exponential. Let $\theta_{X'_i|par(X_i)}$ denote the family of parameters defining the conditional distribution $\Pr(X'_i|par(X_i))$.

Two issues need to be handled in factored domains. First, dynamic programming cannot be performed to compute the Q-values at each state in polynomial time. Instead Monte Carlo Value Iteration [12] is used to approximate Q-values at a sample of reachable states. Second, even though the same state is not revisited in any trajectory, the same transition parameters will be used at each stage of the process. So instead of partitioning the parameters by levels, they are partitioned by families corresponding to different conditional distributions. This allows alternation between a sequence of linear programs as before.

The Monte Carlo Value Iteration technique, originally designed for continuous POMDPs [12], is adapted for use with factored discrete MDPs. Instead of storing an exponentially large Q-function at each stage, a policy graph $G = \langle N, E \rangle$ is now stored. The nodes $n \in N$ of policy graphs are labeled with actions, and the edges $e \in E$ are labeled with observations (i.e., values for the test corresponding to the previous action). A policy graph $G = \langle \phi, \psi \rangle$ is parameterized by a mapping $\phi : N \to A$ from nodes to actions and a mapping $\psi : E \to N$ from edges to next nodes. Since each edge is rooted at a node and labeled with an observation, ψ is also referred to as a mapping from node-observation pairs to next nodes (i.e. $\psi : N \times O \to N$). Here an observation is the result of a test. A useful operation on policy graphs will be to determine the best value that can be achieved at a given state by starting in any node. Algorithms 5 describes how to compute this by Monte Carlo sampling. k trajectories are sampled starting in each node. The node with the highest value is returned along with its value.

The main purpose of the policy graph is to provide a succinct and implicit representation of a value function. More precisely, the value of a state can be estimated by calling evalGraph(G, s). While the policy graph could also be used as a controller, instead it is used to perform a one step look ahead to infer the best action to execute at each step in the same way that it would be done if an explicit value function was available to extract a policy. In other words, given a value function V, the best action a^* for any state s can be extracted by computing

$$a^* = \arg\max_a \rho(s, a) + \gamma \sum_{s'} \Pr(s'|s, a) V(s')$$

Similarly, the best action to execute at each time step when in state s can also be extracted

Algorithm 5 Evaluate G at s

evalGraph(G, s)

```
1  Let N be the set of nodes for G = ⟨φ, ψ⟩
2  for each n ∈ N
3        V(n) ← 0
4        repeat k times
5              V(n) ← V(n) + evalTrajectory(G, s, n)/k
6  n* ← argmax_{n∈N} V(n)
7  return V(n*) and n*
```

evalTrajectory(G, s, n)

```
8  Let G = ⟨φ, ψ⟩
9  if n does not have any edge
10       return ρ(s, φ(n))
11 else
12       Sample o ~ Pr(o|s, φ(n))
13       Let s' be the state reached when observing o after executing φ(n) in s
14       return ρ(s, φ(n)) + γ evalTrajectory(G, s', ψ(n, o))
```

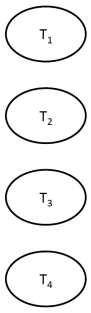


Figure 5.1: Sample Policy Graph after 1 iteration of Algorithm 6

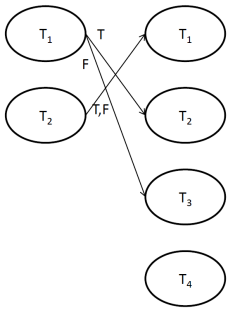


Figure 5.2: Sample Policy Graph after 2 iterations of Algorithm 6

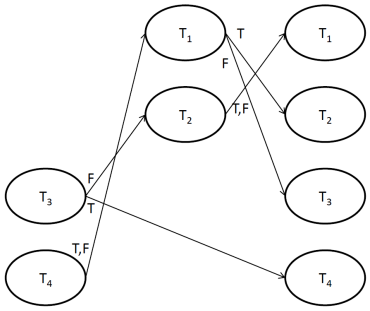


Figure 5.3: Sample Policy Graph after 3 iterations of Algorithm 6

based on policy graph G by computing

$$a^* = \arg\max_a \rho(s,a) + \gamma \sum_{s'} \Pr(s'|s,a)\textsc{evalGraph}(G,s')$$

Algorithm 6 describes how to construct a policy graph G by approximate value iteration. Here, value iteration is performed by approximate backups that compute and store a policy graph instead of a value function at each step. Figures 5.1, 5.2, and 5.3 present a sample trace of how the policy graph may appear after each iteration of the for loop in Algorithm 6 on line 2. Initially, all actions are present as disconnected nodes. As more iterations are completed, more nodes are added to the graph. Each node represents an action and each arrow represents the observation obtained after executing that action. The arrow links to another node that indicates the next action to execute after an observation for a given action.

The graph is constructed by performing point-based backups only at a set of states *setOfStates*. This set of states can be obtained in several ways. It should be representative of the reachable states and allow for the construction of a good set of conditional plans. As it will be discussed later, it is desirable to include in *setOfStates* all the states s' that are reachable from the states s for which constraints $\langle s, a^* \rangle$ area available. At each iteration, a new node is added to the policy graph for each state in *setOfStates*. Although not shown in Algorithm 6, redundant nodes could be pruned from the policy graph to improve efficiency.

Similar to flat MDPs, the parameters of the conditional distributions need to be optimized to satisfy the expert's constraints. The Q-values on which we have constraints can be approximated by the evalGraph procedure.

$$Q^G(s,a) = \rho(s,a) + \gamma \sum_{s'} \Pr(s'|s,a)\textsc{evalGraph}(\text{G,s'}) \tag{5.9}$$

Since the Q-function has a non-linear dependence on the transition parameters, the parameters are partitioned in families $\theta_{X_i'|par(X_i)}$ corresponding to conditional distributions $\Pr(X_i'|par(X_i))$ for each test variable X_i with the corresponding action a_i that selects to observe X_i. Alternating between the optimization of different families of parameters ensures that the optimization is linear. In any trajectory, a variable X_i is observed at most once and therefore at most one transition parameter for the observation of X_i participates in the product of probabilities of the entire state trajectory. Hence, the Q function can be written as a linear combination of the parameters of a given family

$$Q(s,a) = c(nil) + \sum_{o,x} c(o,x) \Pr(X_i' = o|par(X_i) = x) \tag{5.10}$$

Algorithm 6 Monte Carlo Value Iteration

MCVI($setOfStates, horizon$)

```
1  Initialize G with no edge and |A| nodes such that φ maps each node to a different action
2  for t = 1 to horizon
3      for each s ∈ setOfStates
4          for each a ∈ A
5              Q(s, a) ← ρ(s, a)
6              for each o observable from s after executing a
7                  Let s' be the state reached when observing o after executing a in s
8                  [V(s'), n_{a,o}] ← EVALGRAPH(G, s')
9                  Q(s, a) ← Q(s, a) + γ Pr(o|s, a) V(s')
10         a* ← argmax_a Q(s, a)
11         Add new node n to G such that φ(n) = a* and ψ(n, o) = n_{a*,o}
12 return G
```

where $c(nil)$ denotes a constant and $c(o, x)$ is the coefficient of the probability of observing outcome o for X'_i given that the joint value of the parent variables of X'_i is x. Algorithm 7 shows how to compute the linear dependency on the parameters of $\Pr(X'_i|par(X_i))$. More precisely, it computes a vector c of coefficients by sampling k trajectories in G and averaging the linear coefficients of those trajectories. In each trajectory, a recursive procedure computes the coefficients based on three cases: i) when n is a leaf node (i.e., no edges), it returns the reward as a constant in $c(nil)$; ii) when a is executed for the first time, it returns the value of each o in $c(o, x)$; iii) otherwise, it recursively calls itself and adds the reward in $c(nil)$.

Similar to the linear program (5.8) for flat MDPs, a linear program can be defined to optimize the transition parameters of a single family subject to linear constraints on Q-values as defined in Equation 5.10. It is also possible to alternate between the optimization of different families similar to Algorithm 4, but for factored MDPs.

5.4 Evaluation and Experiments

Formally, for M^*, the true MDP that needs to be learnt, the optimal policy π^* determines the choice of best next test as the one with the highest value function. If the correct choice

Algorithm 7 Linear dependency of $Q^G(s,a)$ on parameters of $\Pr(X_i'|par(X_i))$ when executing a in s and following G thereon. This function returns the coefficients c of $\Pr(X_i'|par(X_i))$ based on k sampled trajectories of G.

linearDependence(G, s, a, i)

1 $c(nil) \leftarrow \rho(s,a)$ and $c(o,x) \leftarrow 0\ \forall o \in dom(X_i'), x \in dom(par(X_i))$
2 **repeat** k times
3 Sample s' from $\Pr(s'|s,a)$
4 Let n' be the node created in G for $s' \in setOfStates$
5 $c \leftarrow c + \gamma$linearDependenceRecursive$(G, s', n', i)/k$
6 **return** c

linearDependenceRecursive(G, s, n, i)

7 **if** n does not have any edge
8 $c(nil) \leftarrow \rho(s, \phi(n))$
9 $c(o,x) \leftarrow 0\ \forall o \in dom(X_i'), x \in dom(par(X_i'))$
10 **else if** $\phi(n) = a_i$ and $\phi(n)$ is executed for the first time
11 $c(nil) \leftarrow 0$
12 Let x be the part of s referring to $par(X_i)$
13 $c(o,x') \leftarrow 0\ \forall o \in dom(X_i), x' \neq x$
14 **for** each o observable when executing $\phi(n)$ in s
15 Let s' be the state reached when observing o after executing $\phi(n)$ in s
16 $c(o,x) =$ evalTrajectory$(G, s', \psi(n,o))$
17 **else**
18 Sample $o \sim \Pr(o|s, \phi(n))$
19 Let s' be the state reached when observing o after executing $\phi(n)$ in s
20 $c \leftarrow \gamma$ linearDependenceRecursive$(G, s', \psi(n,o), i)$
21 $c(nil) \leftarrow \rho(s, \phi(n)) + c(nil)$
22 **return** c

for the next test is known (such as demonstrated by an expert), this information can be used to include a constraint on the model. Let $\mathbf{\Gamma}^+$ denote the set of observed constraints and $\mathbf{\Gamma}^*$ denote the set of all possible constraints that hold for M. Having only observed $\mathbf{\Gamma}^+$, the refinement technique discussed in this chapter will consider any $M^+ \in \mathbf{M}^+$ as a possible true model, where $\mathbf{M}^+$ is the set of all models that obey $\mathbf{\Gamma}^+$. Let $\mathbf{M}^*$ denote the set of all models that are *constraint equivalent* to M^* (i.e., obey $\mathbf{\Gamma}^*$) and $\tilde{M}$ denote the initial model, and $\hat{M}_{\mathbf{\Gamma}^+}$ denote the particular model obtained by iterative model refinement based on the constraints $\mathbf{\Gamma}^+$.

Ideally, the goal is to find the true underlying model M^*, hence the KL-divergence$(M^*, \hat{M}_{\mathbf{\Gamma}^+})$ is reported. However, other constraint equivalent models may recommend the same actions as M^* and thus have similar constraints, so *test consistency* with $\mathbf{M}^*$ (i.e., # of states in which optimal actions are the same) and the *simulated value of policy* of $\hat{M}_{\mathbf{\Gamma}^+}$ with respect to the true transition function T are two other metrics by which the refined model is also evaluated.

Given a consistent set of constraints $\mathbf{\Gamma}$ and sufficient time (for random restarts to find the global optimum), this technique for model refinement will choose a model $\hat{M}_{\mathbf{\Gamma}} \in \mathbf{M}^*$ by construction. If the constraints specified by the expert are inconsistent (i.e., do not correspond to any possible model), the approach minimizes the violation of the constraints as much as possible through alternating optimization combined with random restarts. The best solution found is reported after exhausting the time quota to perform refinement.

5.4.1 Experimental Results on Synthetic Problems

This section describes the results on a 4-test recommender system. The objective is to discover the transition model of some model $M^* \in \mathbf{M}^*$. M^* is selected by randomly sampling its transition and reward functions. Given this model M^*, a set of constraints $\mathbf{\Gamma}^+$ are sampled and the refinement technique is used to find $\hat{M}_{\mathbf{\Gamma}^+}$. To evaluate $\hat{M}_{\mathbf{\Gamma}^+}$, first the constraints $\mathbf{\Gamma}^*$ for M^* are computed, then the set of constraint-equivalent models $\mathbf{M}^*$ is estimated by sampling 100 models from $\mathbf{M}^*$. These constraint equivalent models are then compared with $\hat{M}_{\mathbf{\Gamma}^+}$.

The KL-divergence between each constraint-equivalent model and the refined model KL-DIV$(M_i^*, \hat{M}_{\mathbf{\Gamma}^+})$ is computed. Its ratio with the KL-divergence between the constraint equivalent model and the initial model KL-DIV$(M_i^*, \tilde{M})$ is shown in Figure 5.4. A lower value of this ratio indicates that the refined model $\hat{M}_{\mathbf{\Gamma}^+}$ is closer to the true model M^* than the initial model $\tilde{M}$. It can be seen that the mean KL-divergence decreases as the number of constraints in $\mathbf{\Gamma}^+$ increases since the feasible region becomes smaller. Figures 5.5 and 5.6

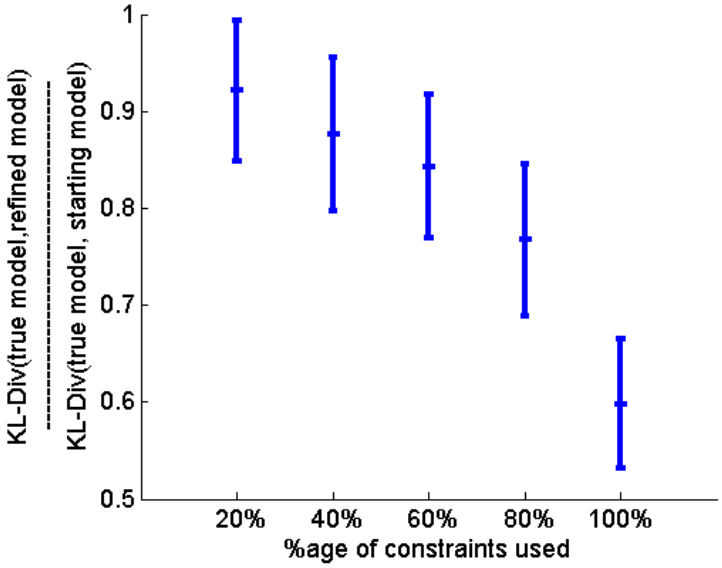


Figure 5.4: Ratio of KL-divergence of True Model with Refined Model after Model Refinement for MDPs to True Model with Initial Model – Synthetic Problem

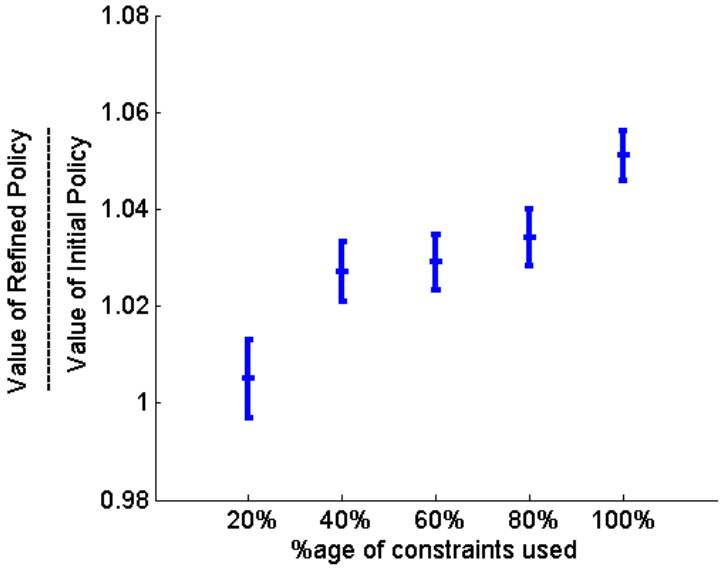


Figure 5.5: Ratio of Refined Policy Value after Model Refinement for MDPs to Initial Policy Value – Synthetic Problem

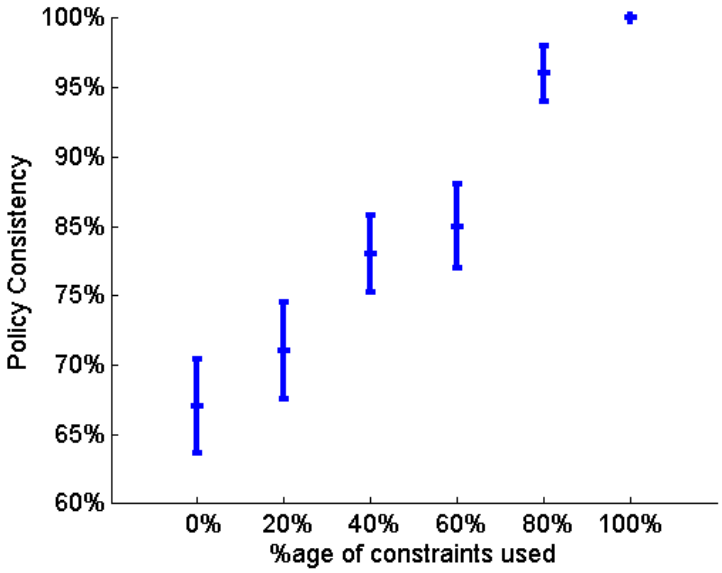


Figure 5.6: Policy Consistency after Model Refinement for MDPs – Synthetic Problem

show similar trends for policy consistency and simulated value of the policy. Similar trends were also observed for KL-divergence, policy consistency and simulated value of policy with recommender system models with more than 4 variables.

5.4.2 Experimental Results on Diagnostic Problems

The refinement technique is evaluated specifically for diagnostic MDPs. To construct such MDPs, the number of tests and causes are chosen. The total actions in the MDP are the sum of the tests and causes with an action either being the option to execute a test and observe its value or make a diagnostic prediction regarding the cause. Executing a test has a small negative reward. The diagnostic prediction has a high positive reward if the correct cause is diagnosed and a high negative reward for an incorrect diagnosis. No discount factor is used as it is a finite horizon problem.

Diagnostic MDPs are better represented as factored MDPs since executing a test only affects a part of the state space, i.e., the probabilities of other tests that are similar in nature. While small diagnostic MDPs can be encoded with a flat representation, a factored representation allows a more succinct representation with fewer parameters to be learned for the transition function.

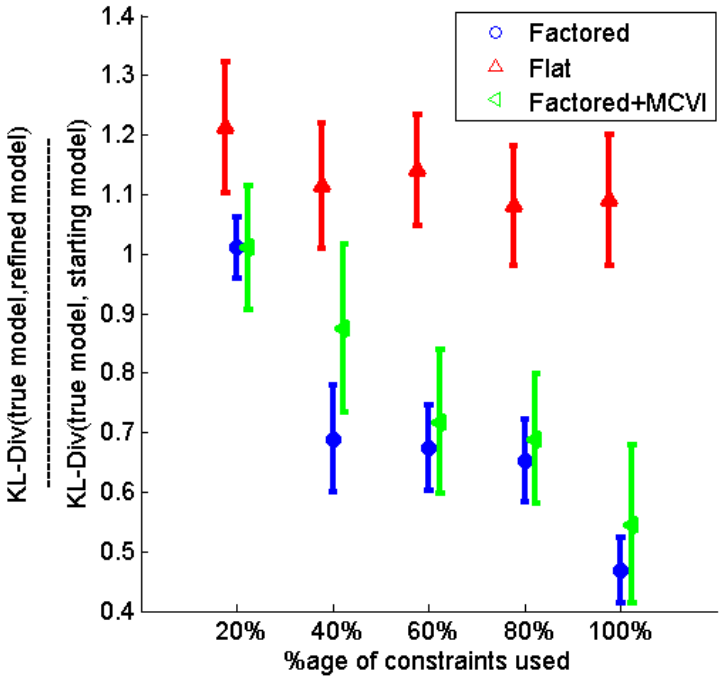


Figure 5.7: Ratio of KL-divergence of True Model with Refined Model after Model Refinement for MDPs to True Model with Initial Model – Diagnostic Problem

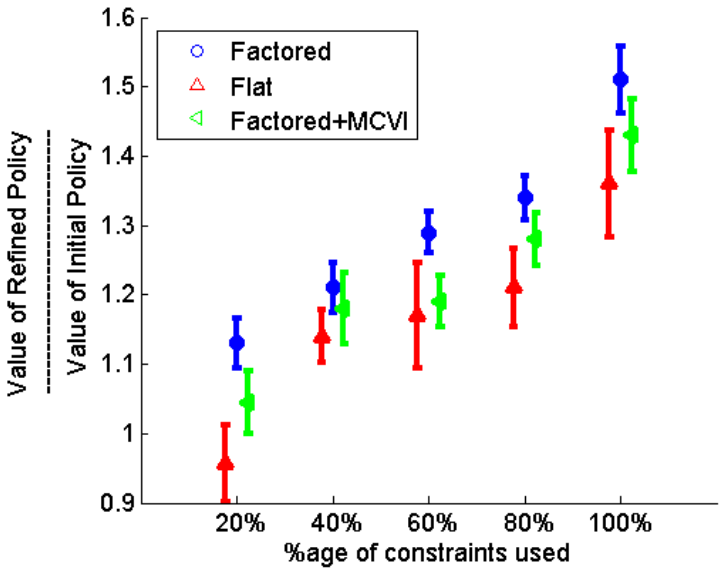


Figure 5.8: Ratio of Refined Policy Value after Model Refinement for MDPs to Initial Policy Value – Diagnostic Problem

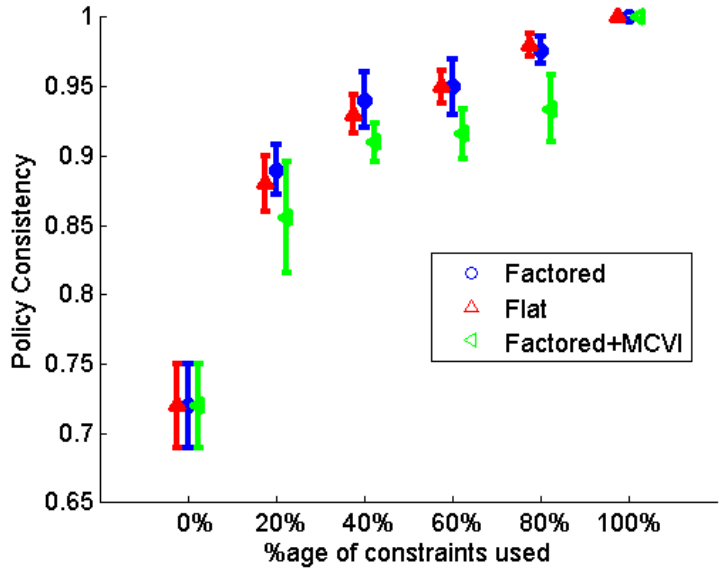


Figure 5.9: Policy Consistency after Model Refinement for MDPs – Diagnostic Problem

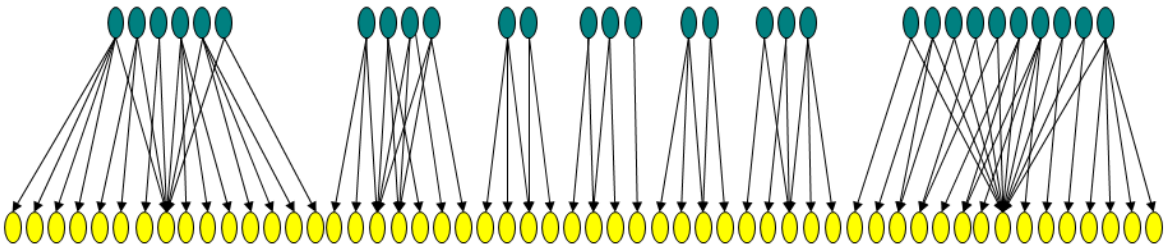


Figure 5.10: Large Scale Diagnostic Bayesian Networks converted to MDP

The results of model refinement are presented on the same diagnostic MDP represented as a flat MDP, a factored MDP with exact value iteration and a factored MDP with Monte Carlo Value Iteration (MCVI) in Figures 5.7, 5.8, and 5.9. These results are shown for a 4-cause and 4-test network. It can be observed that the factored representation yields better results than the flat representation. This is because the factored representation exploits the inherent structure of the diagnostic MDP, whereas the flat representation is unable to preserve this structure after refinement. This is clearly evident in the case of KL-divergence where the resulting model does obey the constraints, but is in fact farther away from the true model than the starting model. It can also be seen that considering a subset of states for $setOfStates$ in MCVI (states reachable from constraints with 50% of remaining states), the results for KL-divergence, test consistency and value of policy deteriorate in comparison to the exact factored case. In separate experiments, it is observed that increasing the size of $setOfStates$ results in improved refined models and decreasing them results in refined models that are not as good. For the purpose of this work, MCVI is only being used as a method to solve factored MDPs and demonstrate the technique for refinement on a large problem. The question of determining an optimal $setOfStates$ for MCVI is not considered in this thesis, though this question has been extensively studied in point-based value iteration algorithms for POMDPs [87].

5.4.3 Experimental Results on Large Scale Diagnostic Problems

The refinement technique is also evaluated in this section on real-world diagnostic networks collected and reported by Agosta et al. [6]. These networks, also described in Section 4.4.3, were collected through detailed session logs over a period of seven weeks in which the entire diagnostic sequence was recorded. The sequences intermingle model building and querying

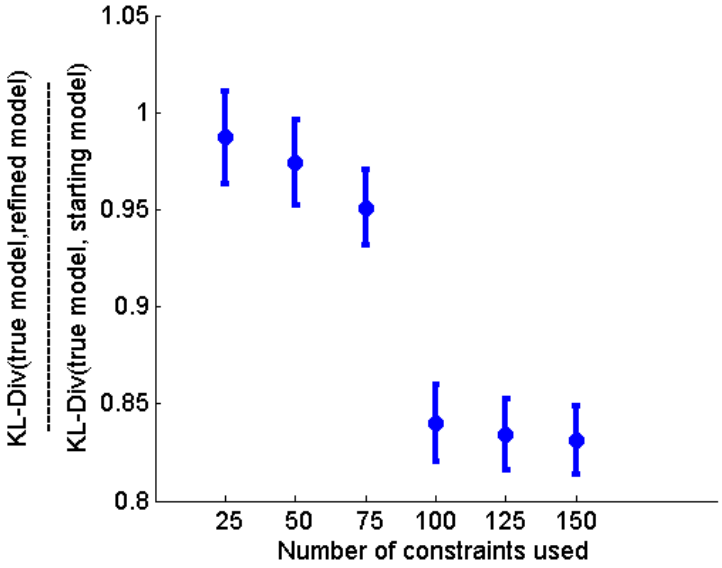


Figure 5.11: Ratio of KL-divergence of True Model with Refined Model after Model Refinement for MDPs to True Model with Initial Model – Large Scale Diagnostic Problem

phases. The model network structure was inferred from an expert's sequence of positing causes and tests. Test-ranking constraints were deduced from the expert's test query sequences once the network structure is established.

The logs captured 157 sessions over seven weeks that resulted in a model with 115 tests and 82 root causes. The network consists of several disconnected sub-networks, each identified with a symptom represented by the first test in the sequence, and all subsequent tests applied within the same subnet. There were 20 sessions in which more than two tests were executed, resulting in a total of 32 test constraints. The diagnostic network is pruned to remove the sub-networks with no constraints to get 54 tests and 30 root causes, divided in 7 sub-networks, as shown in Figure 5.10. The model refinement technique is applied to learn the parameters for each sub-network separately. The largest sub-network has 15 tests and 10 causes resulting in 25 possible actions and more than 14 million states. MCVI is used for larger networks of this scale as it would not be possible to solve them exactly otherwise.

The 32 constraints extracted from the session logs are used to represent a feasible region from which 100 true models are sampled. 1000 states are sampled in addition to the states reachable by the constraints to form the $setOfStates$ used by MCVI. The approximation

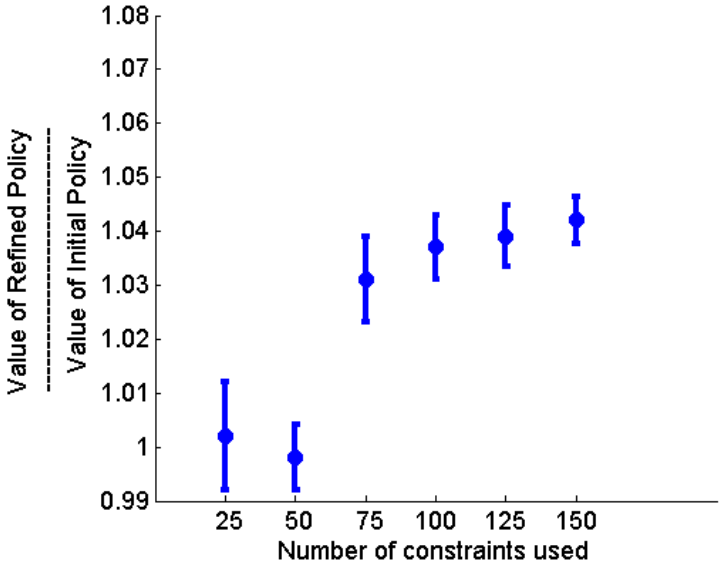


Figure 5.12: Ratio of Refined Policy Value after Model Refinement for MDPs to Initial Policy Value – Large Scale Diagnostic Problem

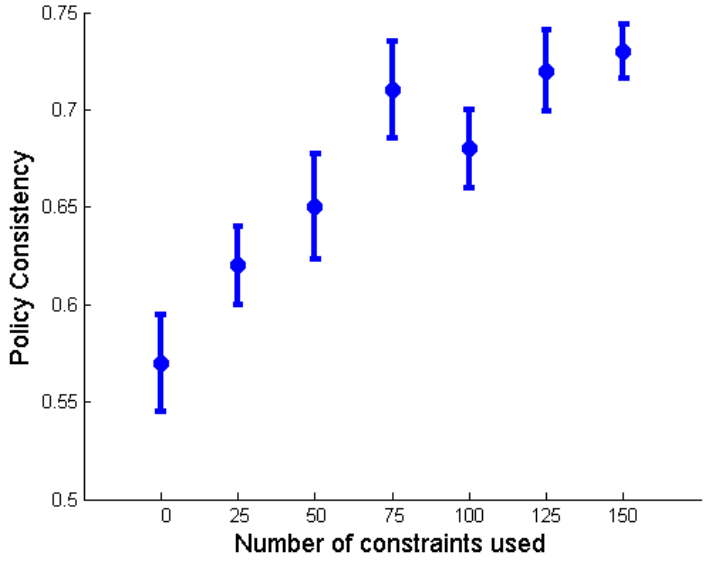


Figure 5.13: Policy Consistency after Model Refinement for MDPs – Large Scale Diagnostic Problem

in MCVI can result in situations where no feasible model is available during refinement. In such a case, the experiments are stopped after an allocated amount of time and the model that violates the constraints the least among those computed so far is reported. For the experiments in this section, the refinement process was terminated after 10 random restarts of the alternating optimization problem, i.e., randomly perturbing the parameters 10 times after the solution had locally converged before choosing the best solution available till that time.

Figures 5.11, 5.12, and 5.13 show the results for KL-divergence, simulated value of policy and policy consistency respectively for the real world diagnostic network. Since the total number of constraints is exponential, a subset of constraints are randomly sampled and the results are shown using these subsets instead of a percentage of all possible constraints. Similarly, the policy consistency is also computed by randomly sampling 100 states and then comparing optimal actions in those states, which also explains the high standard deviation. It can be seen that using a small subset of constraints and a small number of states as input to MCVI yields benefits in moving closer to the original model.

5.5 Summary

The computation of optimal policies for MDPs involves complicated numerical calculations. If the policy is incorrect and it has been critiqued by an expert, the process of updating the model to ensure the prevention of these mistakes is non-trivial for human beings. This chapter presents an approach to refine the transition function of an MDP based on feedback from an expert.

Several approaches have been proposed to refine reward functions based on inverse reinforcement learning with the assumption that the transition function is known. Refining the reward function given a known transition function can be posed as a linear optimization problem. However, the problem of learning the transition function given the reward function is non-convex and has not been addressed. This chapter presents an approach to incorporate these non-convex constraints implied from expert feedback on policies in the process of refining the transition function. The three major important contributions of this work are as follows. First, the problem of defining constraints on the parameters of the transition function using feedback from an expert is formalized. This feedback may be implicit when obtained from logs of diagnostic sessions performed by a domain expert. Second, an approach to handle these non-convex constraints is provided for the flat and factored version of an MDP. Third, the scalability of this approach is demonstrated in the domain of diagnostics by adapting the technique of approximate Monte Carlo value

iteration from large-scale POMDPs to large-scale MDPs. The results presented in this chapter indicate that the refinement technique not only helps in getting closer to the true transition function, but also improves policy consistency and the value of the policy.

The technique for refining MDPs presented in this chapter is demonstrated for factored MDPs for which the optimal policy is calculated using Monte Carlo value iteration and policy graphs. Other exact or approximate approaches to calculate the optimal policy for a factored MDP can be used without affecting the refinement technique. The only change will be the process through which the value function for future states (EvalGraph in 5.9) is calculated.

Chapter 6

Conclusion

Decision-theoretic planning provides a principled approach to computing policies in environments with stochasticity as well as account for complex objectives. There are two key challenges that are preventing the widespread adoption of such approaches. First is the lack of any mechanism to explain the policies or otherwise validate their correctness. Second is the lack of methods to automatically refine these models by observing expert behavior. This thesis deals with both of these problems.

6.1 Summary of Contributions

Chapter 3 presents an approach to explain policies for MDPs. These explanations are presented by populating templates that report the occupancy frequencies of states with high/low reward as they are the states which the users prefer/dislike. For factored MDPs, instead of states the occupancy frequencies for scenarios are used. Sample explanations for course advising and hand washing domains are presented. A user study is also conducted to evaluate explanations in course advising. Feedback from students indicates that the explanations are well-received as a supplementary source of knowledge in addition to discussions with human advisor regarding the optimal set of courses.

Chapter 4 presents a technique to incorporate expert feedback in model refinement of Bayesian network parameters. The feedback received from an expert is the preference of one action over another in a given state. This feedback is translated as a constraint using Gini index to determine which action should have the lowest value for Gini index. Since this constraint is non-convex, Gibbs sampling is used to search for the MAP estimate

for the parameters of the Bayesian network that satisfy the test orderings obtained from the expert. The first approach is based on rejection sampling to reject any samples that violate these constraints. The second approach does not require rejection sampling but converges to the same MAP for the parameters in much less time. Furthermore, if any data is available, it can also be used as a part of the learning process. The KL-divergence and test consistency of the refined models that use constraints are shown to be better than the models refined only using data for diagnostic Bayesian networks drawn from real-world manufacturing domains.

Chapter 5 extends the notion of incorporating expert feedback in model refinement to MDPs. The expert feedback again translates into a constraint that the value function of the action recommended by the expert should not be less than the value function for any other action. This constraint is again non-convex for the case of known reward function and unknown transition function. The parameters of the transition function are partitioned in subsets such that each subset can then be optimized iteratively through linear optimization while fixing the values of the parameters in the other subsets. To demonstrate the scalability of this approach for large-scale MDPs, the technique of policy graphs with Monte Carlo value iteration for POMDPs is adapted for use with MDPs. Under this setting, an approach to refine the model parameters is presented. The results of refinement of MDP model parameters are evaluated for KL-divergence, policy consistency and expected value of policy. It is observed that the use of constraints results in improvement in all these criteria when constraints are used.

Policy explanation should help recommender systems based on MDPs gain wider acceptance as it will be easier to understand the policy better and gain user trust as a consequence. Model refinement based on user constraints allows incorporating expert feedback into the model to avoid repetition of similar mistakes in future. It also allows harnessing an important source of information since it is difficult to obtain lots of data to construct accurate models when each data point requires the participation of a user.

6.2 Directions for Future Research

The work presented in this thesis can be extended in various directions. This section briefly discusses some of these research directions.

6.2.1 Extensions for Policy Explanation

- The current approach for explanations is based on populating pre-defined templates. Explanations for end-users without any knowledge about probability theory or utility theory may require more natural language generation or further abstraction by even removing the notion of occupancy frequencies. Subsequent to the work on explanations presented in this thesis, Dodson et al., [25, 26] in fact have presented an approach for explanations of MDP policies based on natural language generation. Other possibilities, such as visually depicting sequences of actions and their relation to goal states may also be useful for users to understand the consequences of different choices.

- The work on explaining MDPs will be interesting to extend for the case of POMDPs. Since the states are not directly observable in POMDPs, it is not obvious how one could generate an explanation that refers to the frequency with which some states are visited. In fact, another aspect of the explanation for the POMDP may involve providing some information regarding the current belief state.

- It would also be interesting to extend the work on explanations to reinforcement learning where the parameters of the model (i.e., transition function and reward function) are unknown or partially known. This may require incorporating some ideas from robust MDPs and imprecise parameter MDPs to explanations or otherwise evaluate the sensitivity of the explanation to the uncertainty of the model.

6.2.2 Extensions for Model Refinement

- The work on refining parameters for a MDP assumes a known reward function and unknown transition function. The natural extension to this will be the case where both the transition and reward functions are unknown or can be refined. It is not clear if the alternating optimization approach can be adapted when the reward function is also included as a variable in the optimization problem.

- Another possible extension for model refinement is to consider the case of POMDPs and also refine the observation function in conjunction with the transition function. Again it is not trivial to extend the alternating optimization technique as the constraint will include a product of the transition function with the observation function. Another issue with the extension of this work to POMDPs is a more fundamental question of formally representing the constraint. Since the state is not observable

so some mechanism will need to be formulated to elicit the expert's belief about the current state on which the constraint is being formulated.

References

[1] Merriam-Webster.com, Retrieved on 2013-06-19 from http://www.merriam-webster.com/dictionary/explain.

[2] Agnar Aamodt and Enric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Commuication*, 7(1):39–59, 1994.

[3] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-First International Conference on Machine Learning (ICML)*, Banff, AB Canada, 2004.

[4] Pieter Abbeel and Andrew Y. Ng. Exploration and apprenticeship learning in reinforcement learning. In *Proceedings of the Twenty Second International Conference on Machine Learning (ICML)*, Bonn, Germany, August 2005.

[5] Pieter Abbeel, Morgan Quigley, and Andrew Y. Ng. Using inaccurate models in reinforcement learning. In *Proceedings of the Twenty Third International Conference on Machine Learning (ICML)*, Pittsburgh, PA, USA, 2006.

[6] John Mark Agosta, Omar Zia Khan, and Pascal Poupart. Evaluation results for a query-based diagnostics application. In *Fifth Workshop on Probabilistic Graphical Models (PGM)*, Helsinki, Finland, 2010.

[7] Gennady Agre. Diagnostic Bayesian networks. *Computers and Artificial Intelligence*, 16(1), 1997.

[8] Eric E. Altendorf, Angelo C. Restificar, and Thomas G. Dietterich. Learning from sparse data by exploiting monotonicity constraints. In *Proceedings of Twenty First Conference on Uncertainty in Artificial Intelligence (UAI)*, Edinburgh, Scotland, July 2005.

[9] Eitan Altman. *Constrained Markov Decision Processes*. CRC Press, 1999.

[10] Brigham S. Anderson and Andrew W. Moore. Fast information value for graphical models. In *Proceedings of the Nineteenth Annual Conference on Neural Information Processing Systems (NIPS)*, pages 51–58, Vancouver, BC, Canada, December 2005.

[11] Amin Atrash and Joelle Pineau. A bayesian reinforcement learning approach for customizing human-robot interfaces. In *Proceedings of the Fourteenth International Conference on Intelligent User Interface (IUI)*, Sanibel Island, USA, February 2009.

[12] Haoyu Bai, David Hsu, Wee Sun Lee, and Vien A Ngo. Monte Carlo value iteration for continuous-state POMDPs. In *Algorithmic Foundations of Robotics IX*. Springer, 2011.

[13] Valentina Bayer-Zubek and Thomas G. Dietterich. Integrating learning from examples into the search for diagnostic policies. *Journal Of Artificial Intelligence Research (JAIR)*, 24:263–303, 2005.

[14] Jennifer Boger, Jesse Hoey, Pascal Poupart, Craig Boutilier, Geoff Fernie, and Alex Mihailidis. A planning system based on Markov decision processes to guide people with dementia through activities of daily living. *IEEE Transactions on Information Technology in Biomedicine*, 10(2):323–333, April 2006.

[15] Craig Boutilier, Thomas Dean, and Steve Hanks. Decision theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research (JAIR)*, 11:1–94, 1999.

[16] Craig Boutilier, Richard Dearden, and Moisés Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1-2):49–107, 2000.

[17] Leo Breiman, Jerome Friedman, Charles J. Stone, and R.A. Olshen. *Classification and Regression Trees*. Chapman and Hall/CRC, 1984.

[18] Urszula Chajewska and Joseph Y. Halpern. Defining explanation in probabilistic systems. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, Providence, RI 1997.

[19] B. Chandrasekaran, Michael C. Tanner, and John R. Josephson. Explaining control strategies in problem solving. *IEEE Expert: Intelligent Systems and Their Applications*, 4(1):9–15, 19–24, 1989.

[20] Richard C. Chen and Eugene A. Feinberg. Non-randomized policies for constrained markov decision processes. *Mathematical Methods of Operations Research*, 66:165–179, 2007.

[21] W. J. Clancey. The epistemology of a rule-based expert system – a framework for explanation. *Artificial Intelligence*, 20:215–251, 1983.

[22] Cassio P. de Campos and Qiang Ji. Improving Bayesian network parameter learning using constraints. In *Proceedings of the Ninth International Conference in Pattern Recognition (ICPR)*, Tampa, FL, USA, 2008.

[23] Erick Delage and Shie Mannor. Percentile optimization in uncertain markov decision processes with application to efficient exploration. In *Proceedings of the Twenty Fourth International Conference on Machine Learning (ICML)*, pages 225–232, New York, NY, USA, 2007. ACM.

[24] Karina Valdivia Delgado, Scott Sanner, and Leliane Nunes de Barros. Efficient solutions to factored MDPs with imprecise transition probabilities. *Artificial Intelligence Journal (AIJ)*, 175:1498–1527, 2011.

[25] Thomas Dodson, Nicholas Mattei, and Judy Goldsmith. A natural language argumentation interface for explanation generation in Markov Decision Processes. In *Proceedings of the Second International Conference on Algorithmic Decision Theory (ADT)*, Piscataway, NJ, 2011.

[26] Thomas Dodson, Nicholas Mattei, Joshua T. Guerin, and Judy Goldsmith. An English-language argumentation interface for explanation generation with Markov Decision Processes in the domain of academic advising. Accepted for publication in:. *ACM Transactions on Intelligent Interactive Systems*, 2013.

[27] Finale Doshi, Joelle Pineau, and Nicholas Roy. Reinforcement learning with limited reinforcement: Using bayes-risk for active learning in POMDPs. In *Proceedings of the Twenty Fifth International Conference on Machine Learning (ICML)*, pages 256–263, Helsinki, Finland, 2008.

[28] Dnal Doyle, Pádraig Cunningham, Derek Bridge, and Yusof Rahman. *Advances in Case-Based Reasoning*, chapter Explanation Oriented Retrieval, pages 157–168. Springer Berlin / Heidelberg, 2004.

[29] Marek J. Druzdzel. Verbal uncertainty expressions: Literature review. Technical Report CMU-EPP-1990-03-02, CMU, May 1989.

[30] Marek J. Druzdzel. Explanation in probabilistic systems: Is it feasible? Will it work? In *Fifth International Workshop on Intelligent Information Systems*, Deblin, Poland, June 2–5 1996.

[31] Marek J. Druzdzel and Linda C. van der Gaag. Elicitation of probabilities for belief networks: combining qualitative and quantitative information. In *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 141–148, Montreal, QC, Canada, 1995.

[32] Francisco Elizalde, Luis EnriqueSucar, Alberto Reyes, and P. deBuen. An MDP approach for explanation generation. In *Workshop on Explanation-Aware Computing (ExaCt) with AAAI-2007*, Vancouver, BC, 2007.

[33] Francisco Elizalde, Enrique Sucar, Manuel Luque, Javier Díez, and Alberto Reyes. Policy explanation in factored Markov Decision Processes. In *Fourth European Workshop on Probabilistic and Graphical Models (PGM)*, Hirtshals, Denmark, 2008.

[34] Francisco Elizalde, Luis Enrique Sucar, Julieta Noguez, and Alberto Reyes. Integrating probabilistic and knowledge-based systems for explanation generation. In *Workshop on Explanation Aware Computing (ExaCt) with ECAI-2008*, Amsterdam, Holland, 2008.

[35] Ad J. Feelders. A new parameter learning method for Bayesian networks with qualitative influences. In *Proceedings of the Twenty Third International Conference on Uncertainty in Artificial Intelligence (UAI)*, Vancouver, BC, July 2007.

[36] P. Gärdenfors. *Knowledge in Flux: Modeling the Dynamics of Epistemic States*. MIT Press, 1988.

[37] Peter Geibel. Reinforcement learning for MDPs with constraints. In *Proceedings of the Tenth European Conference on Machine Learning and Principles of Knowledge Discovery in Databases (ECML-PKDD)*, Berlin, Germany, 2006.

[38] Mara Angeles Gil and Pedro Gil. A procedure to test the suitability of a factor for stratification in estimating diversity. *Applied Mathematics and Computation*, 43(3):221 – 229, 1991.

[39] Robert Givan, Sonia Leach, and Thomas Dean. Bounded-parameter Markov Decision Processes. *Artificial Intelligence*, 122(1-2):71 – 109, 2000.

[40] Shirley Gregor and Izak Benbasat. Explanations from intelligent systems: Theoretical foundations and implications for practice. *MIS Quarterly*, 23(4):497–530, 1999.

[41] Joshua Guerin, Josiah Hanna, Libby Knouse, Nicholas Mattei, and Judy Goldsmith. The academic advising planning domain. In *3rd ICAPS International Planning Competition Workshop*, Sao Paulo, Brazil, 2012.

[42] Xianping Guo and Xinyuan Song. Discounted continuous-time constrained markov decision processes in polish spaces. *The Annals of Applied Probability*, 21:2016–2049, 2011.

[43] David Heckerman and John S. Breese. Causal independence for probability assessment and inference using bayesian networks. *IEEE Systems, Man, and Cybernetics*, 26(6):826–831, November 1996.

[44] David Heckerman, John S. Breese, and Koos Rommelse. Decision-theoretic troubleshooting. *Communications of the ACM*, 38(3):49–56, 1995.

[45] Jonathan L. Herlocker, Joseph A. Konstan, and John Riedl. Explaining collaborative filtering recommendations. In *ACM Conference on Computer Supported Cooperative Work*, pages 241–250, Philadelphia, PA, USA, December 2–6 2000.

[46] Jesse Hoey, Robert St-aubin, Alan Hu, and Craig Boutilier. SPUDD: Stochastic planning using decision diagrams. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 279–288, Stockholm, Sweden, July 30 to August 1 1999.

[47] Jesse Hoey, Axel von Bertoldi, Pascal Poupart, and Alex Mihailidis. Assisting persons with dementia during handwashing using a partially observable Markov decision process. In *Proceedings of the Fifth International Conference of Computer Vision Systems (ICVS)*, Bielefeld, Germany, 2007.

[48] Ronald A. Howard. Information value theory. *IEEE Transactions on Systems Science and Cybernetics*, 2(1):22–26, August 1966.

[49] Ronald A. Howard and James E. Matheson. *Influence Diagrams in Readings on The Principles and Applications of Decision Analysis*. Strategic Decisions Group, 1984.

[50] Omar Zia Khan, Pascal Poupart, and John Mark Agosta. Automated refinement of Bayes networks' parameters based on test ordering constraints. In *Proceedings of the Twenty Fifth Annual Conference on Neural Information Processing Systems (NIPS)*, Granada, Spain, 2011.

[51] Omar Zia Khan, Pascal Poupart, and John Mark Agosta. Iterative model refinement of recommender MDPs based on expert feedback. In *Proceedings of the European Conference on Machine Learning and Principles of Knowledge Discovery in Databases(ECML-PKDD)*, Prague, Czech Republic, 2013.

[52] Omar Zia Khan, Pascal Poupart, and James P. Black. Minimal sufficient explanations for recommendations generated by MDPs. In *Proceedings of Ninteenth International Conference on Automated Planning and Scheduling (ICAPS)*, Thessaloniki, Greece, September 19-23 2009.

[53] Omar Zia Khan, Pascal Poupart, and James P. Black. *Decision Theory Models for Applications in Artificial Intelligence: Concepts and Solutions*, chapter Automatically Generated Explanations for Markov Decision Processes, pages 144–163. IGI Global, 2011.

[54] W. Bradley Knox and Peter Stone. Combining manual feedback with subsequent MDP reward signals for reinforcement learning. In *Proceedings of Ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Toronto, ON Canada, May 2010.

[55] W. Bradley Knox and Peter Stone. Reinforcement learning with human and MDP reward. In *Proceedings of the Eleventh International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Valencia, Spain, June 2012.

[56] Carmen Lacave and Francisco J. Dez. A review of explanation methods for Bayesian networks. *The Knowledge Engineering Review*, 17:107–127, 2002.

[57] Carmen Lacave, Manuel Luque, and Francisco J. Dez. Explanation of Bayesian networks and influence diagrams in Elvira. *IEEE Transactions on Systems, Man, and Cybernetics*, 37(4):952–965, 2007.

[58] Carmen Lacave, Agnieszka Oniko, and Francisco J. Díez. Use of Elvira's explanation facility for debugging probabilistic expert systems. *Knowledge-Based Systems*, 19(8):730–738, 2006.

[59] Percy Liang, Michael I. Jordan, and Dan Klein. Learning from measurements in exponential families. In *Proceedings of Twenty Sixth Annual International Conference on Machine Learning (ICML)*, Montreal, QC, Canada, June 2009.

[60] Wenhui Liao and Qiang Ji. Learning Bayesian network parameters under incomplete data with domain knowledge. *Pattern Recognition*, 42:3046–3056, 2009.

[61] David J. C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.

[62] Yi Mao and Guy Lebanon. Domain knowledge uncertainty and probabilistic parameter constraints. In *Proceedings of Twenty Fifth Conference on Uncertainty in Artificial Intelligence (UAI)*, Montreal, QC, Canada, 2009.

[63] Mausam and Andrey Kolobov. Planning with Markov Decision Processes: An AI perspective. In *Synthesis Lectures on Artificial Intelligence and Machine Learning*. Morgan and Claypool Publishers, 2012.

[64] Deborah L. McGuinness, Alyssa Glass, Michael Wolverton, and Paulo Pinheiro da Silva. Explaining task processing in cognitive assistants that learn. In *Proceedings of AAAI Spring Symposium on Interaction Challenges for Intelligent Assistants*, Palo Alto, CA, 2007.

[65] Ryszard S. Michalski. A theory and methodology of inductive learning. *Artificial Intelligence*, 20:111–116, 1984.

[66] Marvin Minsky. A framework for representing knowledge. Technical Report 306, MIT-AI Laboratory, June 1974.

[67] Michael Montemerlo, Joelle Pineau, Nicholas Roy, Sebastian Thrun, and Vandi Verma. Experiences with a mobile robotic guide for the elderly. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI)*, pages 587–592, Edmonton, AB, Canada, July 28 to August 1 2002.

[68] Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML)*, 2000.

[69] Radu Stefan Niculescu, Tom M. Mitchell, and R. Bharat Rao. Bayesian network learning with parameter constraints. *Journal of Machine Learning Research (JMLR)*, 7:1357–1383, 2006.

[70] Arnab Nilim and Laurent El Ghaoui. Robust control of Markov decision processes with uncertain transition matrices. *Operations Research*, 53(5):780–798, September–October 2005.

[71] Conor Nugent and Pádraig Cunningham. A case-based explanation system for black-box systems. *Artificial Intelligence Review*, 24(2):163–178, 2005.

[72] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1998.

[73] Liam Pedersen, Maria G. Bualat, David Lees, David E. Smith, and Richard Washington. Integrated demonstration of instrument placement, robust execution and contingent planning. In *Proceedings of International Symposium on AI, Robotics and Automation for Space*, Tokyo, Japan, May 2003.

[74] Mark A. Peot and Ross D. Shachter. Learning from what you dont observe. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 439–446, Madison, WI, July 1998.

[75] A.B. Piunovskiy. Dynamic programming in constrained Markov decision processes. *Control and Cybernetics*, 35:645–660, 2006.

[76] Pascal Poupart. *Exploiting Structure to Efficiently Solve Large Scale Partially Observable Markov Decision Processes*. PhD thesis, University of Toronto, 2005.

[77] Bob Price and Craig Boutilier. Implicit imitation in multiagent reinforcement learning. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML)*, Bled, Slovenia, 1999.

[78] Bob Price and Craig Boutilier. Accelerating reinforcement learning through implicit imitation. *Journal of Artificial Intelligence Research (JAIR)*, 19:569–629, 2003.

[79] Martin L. Puterman. *Markov Decision Processes*. Wiley, 1994.

[80] Howard Raiffa. *Decision Analysis: Introductory Lectures on Choices Under Uncertainty*. Addison-Wesley, 1968.

[81] Nathan Ratliff, J. Andrew (Drew) Bagnell, and Martin Zinkevich. Maximum margin planning. In *Proceedings of the Twenty Third International Conference on Machine Learning (ICML)*, Pittsburgh, PA, 2006.

[82] Kevin Regan and Craig Boutilier. Robust policy computation in reward-uncertain MDPs using nondominated policies. In *Proceedings of Twenty-Fourth Conference on Artificial Intelligence (AAAI)*, Atlanta, GA, 2010.

[83] James Royalty, Robert Holland, Judy Goldsmith, and Alex Dekhtyar. POET, the online preference elicitation tool. In *Workshop on Preferences in AI and CP: Symbolic Approaches with AAAI-2002*, Edmonton, AB Canada, 2002.

[84] Stuart Russell. Learning agents for uncertain environments (extended abstract). In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory (COLT)*, pages 101–103, Madison, WI, 1998.

[85] Jay K. Satia and Roy E. Lave Jr. Markovian decision processes with uncertain transition probabilities. *Operations Research*, 21(3):728–740, 1973.

[86] Edlar Shafir, Itamar Simonson, and Amos Tversky. Reason-based choice. *Cogni*, 49:11–36, 1993.

[87] Guy Shani, Joelle Pineau, and Robert Kaplow. A survey of point-based POMDP solvers. *Autonomous Agents and Multi-Agent Systems*, 27:1–51, 2013.

[88] Frode Sørmo, Jörg Cassens, and Agnar Aamodt. Explanation in case-based reasoning—perspectives and goals. *Artificial Intelligence Review*, 24(2):109–143, 2005.

[89] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[90] W. R. Swartout. Xplain: A system for creating and explaining expert consulting programs. *Artificial Intelligence*, 21:285–325, 1983.

[91] William R. Swartout. A digitalis therapy advisor with explanations. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI)*, Cambridge, MA, USA, August 1977.

[92] Lyn M. Van Swo and Janet A. Sniezek. Factors affecting the acceptance of expert advice. *British Journal of Social Psychology*, 44:443–461, 2005.

[93] Ambuj Tewari and Peter L. Bartlett. Bounded parameter markov decision processes with average reward criterion. In *Proceedings of the Twentieth Annual Conference on Learning Theory (COLT)*, pages 263–277, San Diego, CA, 2007.

[94] Nava Tintarev and Judith Masthoff. A survey of explanations in recommender systems. In *Workshop on Recommender Systems & Intelligent User Interfaces with ICDE-2007*, Istanbul, Turkey, 2007.

[95] Matthias C. M. Troffaes. Learning and optimal control of imprecise Markov decision processes by dynamic programming using the imprecise Dirichlet model. In *Soft Methodology and Random Information Systems*, pages 141–148, 2004.

[96] Vladimir Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998.

[97] John von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1953.

[98] Michael P. Wellman. Fundamental concepts of qualitative probabilistic networks. *Artificial Intelligence*, 44(3):257–303, August 1990.

[99] Chelsea C. White III and Hany K. Eldeib. Markov Decision Processes with imprecise transition probabilities. *Operations Research*, 42(4):739–749, 1994.

[100] Michael R. Wick and William B. Thompson. Reconstructive expert system explanation. *Artificial Intelligence*, 54(1-2):33–70, 1992.

[101] Frank Wittig and Anthony Jameson. Exploiting qualitative knowledge in the learning of conditional probabilities of Bayesian networks. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, San Francisco, CA, July 2000.

[102] Huan Xu and Shie Mannor. The robustness-performance tradeoff in Markov decision processes. In *Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems (NIPS)*, Vancouver, BC, Canada, December 4–7 2006.

[103] Nevin Lianwen Zhang and David Poole. A simple approach to Bayesian network computation. In *Proceedings of the Conference on Canadian Artificial Intelligence*, pages 171–178, Banff, AB Canada, 1994.