Iterative Model Refinement of Recommender MDPs based on Expert Feedback

Omar Zia Khan[1], Pascal Poupart[1], and John Mark Agosta[2]**

[1] David R. Cheriton School of Computer Science
University of Waterloo, Ontario, Canada
{ozkhan,ppoupart}@uwaterloo.ca
[2] Toyota InfoTechnology Center, Mountain View, CA, USA
jmagosta@us.toyota-itc.com

Abstract. In this paper, we present a method to iteratively refine the parameters of a Markov Decision Process by leveraging constraints implied from an expert's review of the policy. We impose a constraint on the parameters of the model for every case where the expert's recommendation differs from the recommendation of the policy. We demonstrate that consistency with an expert's feedback leads to non-convex constraints on the model parameters. We refine the parameters of the model, under these constraints, by partitioning the parameter space and iteratively applying alternating optimization. We demonstrate how the approach can be applied to both flat and factored MDPs and present results based on diagnostic sessions from a manufacturing scenario.

1 Introduction

Markov decision processes (MDPs) provide a natural and principled framework for sequential decision making under uncertainty. They are used in a multitude of domains from robotic control to recommender systems. A frequent bottleneck for the deployment of systems based on MDPs is the acquisition of the model i.e., the transition and reward functions. To that effect, reinforcement learning provides numerous approaches to optimize a policy from data (sequences of state-action-reward triples). However, depending on the application, data may be difficult to obtain. For instance, consider the class of recommender systems where the actions recommended by a system are to be executed by a user. Whenever humans are involved in the execution of actions, it is challenging to obtain a significant amount of data because users may be difficult to recruit and each trial can take a while (users may need anywhere from a few seconds to months to execute an action). Furthermore, some application domains such as fault detection/diagnostics offer few cases to collect data since faults are rare events to start with. In other domains, it is also desirable to obtain a good policy before deployment to ensure good performance, but this restricts the amount of data available for training.

In this paper we consider the problem of refining the transition function of a Markov decision process based on user feedback. Such feedback may be implicit by noting the actions followed by an expert during a trial or explicit when an expert directly

** This work was done when the author was associated with Intel Labs confirms or corrects the actions to be executed in some states by inspecting a policy. Such feedback provides valuable information to adjust the transition model of an MDP that may be imprecise due to a lack of data. We formulate the refinement of a transition function as an optimization problem and incorporate expert feedback as constraints. We also show how to exploit certain properties of recommender systems to partition the variables and optimize them in alternation. We demonstrate the approach with a diagnostic scenario in manufacturing.

The paper is structured as follows. Section 2 reviews Markov decision processes and some important properties of recommender systems. Section 3 explains how this work relates to other work. Section 4 describes our approach to refine a transition function based on expert feedback. We first explain how to do this with flat MDPs and then factored MDPs. Section 5 demonstrates the approach for recommender applications with a real-world diagnostic scenario in manufacturing. Finally, Section 6 concludes and suggests some future work.

2 Background

2.1 Markov Decision Processes

A Markov Decision Process (MDP) is defined by the tuple $M = \langle S, A, T, R, \gamma \rangle$ where S is the set of states s, A is the set of actions a, $T : S \times S \times A \to \mathbb{R}$ is the transition function which indicates the probability $\Pr(s'|s,a)$ of reaching s' by executing a in s, $R : S \times S \times A \to \mathbb{R}$ is the reward function which indicates the reward $R(s', s, a)$ of executing a in s and reaching s', γ is the discount factor (value between 0 and 1, with a lower value indicating a greater preference for an immediate reward). Note that we can rewrite the reward function as $R : S \times A \to \mathbb{R}$, where $R(s,a) = \sum_{s' \in S} R(s', s, a) \Pr(s'|s,a)$. We shall use these equivalent notations for the reward function inter-changeably. A policy $\pi : S \to A$ for an MDP provides a mapping from states to actions. Techniques such as value iteration can then be used to compute optimal policies for MDPs in which the Bellman's optimality equation (Eq 1) is used as an update rule and is applied iteratively.

$$V^{\pi^*}(s) = \max_a \left[R(s,a) + \gamma \sum_{s'} \Pr(s'|s,a) V^{\pi^*}(s') \right] \tag{1}$$

$V^{\pi^*}(s)$ denotes the value of executing the optimal policy π^* when starting in state s and is equal to the expected discounted sum of all rewards accumulated by executing it when starting in state s. For a policy to be considered optimal, it means that $V^{\pi^*}(s) \geq V^{\pi}(s) \, \forall s, \pi$. The notation $Q^{\pi}(s,a)$ is used to represent the value of executing a, starting in s and following the policy π from thereon. This can be considered a function that assigns a value to every state-action pair and can be computed using Eq. 2.

$$Q^{\pi}(s,a) = R(s,a) + \gamma \sum_{s' \in S} \Pr(s'|s,a) V^{\pi}(s') \tag{2}$$

In practice, the state space of many MDPs is defined by the cross product of the domain of several variables (or features). Such MDPs are often referred to as factored MDPs since the transition function is the product of several factors, each corresponding to the conditional distribution of a variable given its parents. Optimizing the policy of a factored MDP is notoriously difficult due to the exponential number of states corresponding to all possible joint assignments of the state variables. In this work, we will adapt the Monte-Carlo Value Iteration algorithm [5] (originally developed for POMDPs) to factored MDPs. The key idea in this work is the observation that value iteration implicitly builds a policy graph. Hence, instead of representing the value function over exponentially many states, a policy graph is incrementally constructed. The value function of a policy graph can be approximately evaluated at a given state by Monte Carlo sampling. Hence, approximate value iteration is performed by incrementally constructing a policy graph that provides a sufficient and compact representation from which the value function can be reconstructed.[3]

2.2 MDPs for Recommender Systems

In this work, we focus on recommender systems where an MDP recommends an action to a user 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 each of the possible actions to record the value, i.e., tests and grades in courses. The actions are recommendations for the next diagnostic test or the next course to register. Furthermore, we assume that repeating an action does not change the result.

Figure 1 presents the flat representation of a toy diagnostic MDP with three state variables and four actions. 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, there are two test variables with domain $\{T, F, \_\}$ and one cause variable with domain $\{C_1, C_2, \_\}$ where the value $\_$ indicates that the variable has not been observed yet. The cause variable records the cause identified by the decision maker (if any) instead of the true underlying cause. We do not use any variable to encode the underlying cause since the test variables already encode all the information that would normally be used to express a distribution over the underlying cause. The actions consist of performing one of the tests or identifying a cause. 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 $\_$.

The states can be organized in levels, where each level groups all the states with the same number of variables instantiated. For instance, at level 0, no variable has been observed and only 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. We shall use the concept of levels to enforce

[3] Although the algorithm builds a policy graph, it is not a policy iteration algorithm, but definitely an approximate form of value iteration since the policy graph only serves as a compact representation from which the value function can be evaluated.

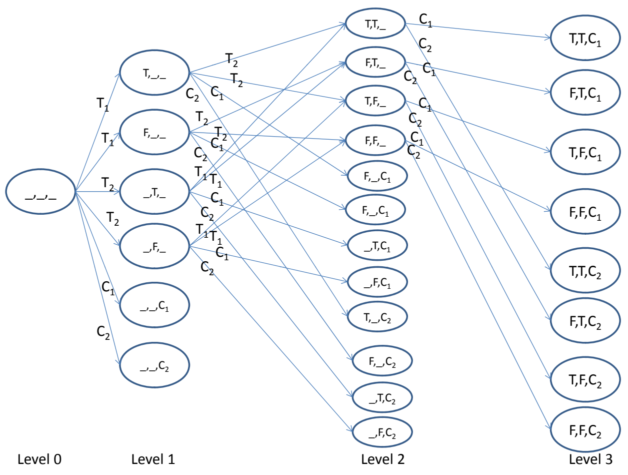

Fig. 1. Sample Flat Recommender MDP

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.

2.3 Problem Statement

The expert designs the MDP by defining the state variables, the actions, and then estimating/specifying transition and reward functions, as well as a discount factor. In this paper, we assume the reward function R and the discount factor γ are specified accurately, while the transition function is imprecise. Let us denote the imprecise transition function as $\tilde{T}$ and the resulting imprecise MDP as $\tilde{M} = \langle S, A, \tilde{T}, R, \gamma \rangle$. Let the true underlying MDP be denoted as $M = \langle S, A, T, R, \gamma \rangle$, where T is the actual transition function. Since $\tilde{T}$ is imprecise, the optimal policy for $\tilde{M}$, $\tilde{\pi}^*$, may also not be truly optimal, *i.e.,* we are not guaranteed that $\tilde{\pi}^* = \pi^*$. 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. Our 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 $\hat{M} = \langle S, A, \hat{T}, R, \gamma \rangle$ obeys all constraints and matches the true optimal policy for these states, *i.e.*, $\hat{\pi}^*(s) = \pi^*(s)$.

3 Related Work

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 learn reward function or otherwise learn the optimal policy without learning the reward function. Inverse reinforcement learning deals with recovering a reward function using a known policy and transition function [9]. In imitation learning [11], the goal is to learn a policy as good as demonstrated by the expert. In apprenticeship learning [1], the expert demonstrations are considered as parts of the optimal policy that would be obtained using the unknown true reward function. Imitation learning 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 [12]. 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 [13].

The above approaches exploit additional information from the expert while assuming a known transition function and unknown reward function. The problem of learning a reward function when the transition function is fixed can be posed as a linear optimization problem. Our objective is to learn the transition function while assuming a known reward function and expert feedback. Estimating the transition function based on constraints implied by user feedback leads to a non-linear non-convex optimization problem. There has been prior work on learning Bayes' nets when using additional knowledge in the form of constraints that are linear [10], convex [6], and non-convex [8]. However, in Bayes' nets the constraints only provide information about the immediate action whereas in MDPs, the policies are sequential in nature and need to account for possible future plans. Constrained reinforcement learning [7] and constrained MDPs [4] have been proposed to handle multi-objective scenarios, but the constraints in these cases are often of the form which limit the value of a policy. In our case, the constraints that arise from expert feedback are imposed on the Q function instead of the policy which makes the problem non-convex and harder to solve. Abbeel and Ng [2] 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 using a maximum likelihood technique on these state-action trajectories to estimate the transition function. Such approaches assume the availability of significant feedback from experts which may be fine for control problems in robotics but not for cases where feedback from expert is very limited (such as diagnostics).

4 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.

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

We explain how to refine the transition model based on such constraints for "flat" MDPs (Section 4.1) and then for "factored" MDPs (Section 4.2).

4.1 Flat Model Refinement

We can setup an optimization problem 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.

$$\max_{\hat{T},\delta} \; \delta \quad \text{s.t.} \quad Q^{\hat{\pi}}(s,a^*) \geq Q^{\hat{\pi}}(s,a) + \delta \quad \forall \langle s,a^* \rangle \in \Gamma, \forall a \tag{3}$$

When δ is non-negative, the refined model satisfies the expert's constraints. If the user's constraints are inconsistent, we will simply find a model that minimizes the degree of violation for all constraints. For problems with a finite horizon h, we can rewrite the Q function as a sum of expected rewards

$$Q^{\pi}(s_0,a_0) = R(s_0,a_0) + \sum_{t=1}^{h} \gamma^t \sum_{s_t} \Pr(s_t|s_0,a_0,\pi) R(s_t,\pi(s_t)) \tag{4}$$

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

$$\Pr(s_t|s_0,a_0,\pi) = \sum_{s_{1..t-1}} \Pr(s_1|s_0,a_0) \prod_{i=2}^{t} \Pr(s_i|s_{i-1},\pi(s_{i-1})) \tag{5}$$

In a flat MDP, the transition probabilities are the transition parameters. Hence, we will denote by θ the vector of transition parameters $\theta_{s'|s,a} = \Pr(s'|s,a)$. We can then rewrite the optimization problem (3) in terms of θ by substituting Equations 4 and 5:

$$\max_{\hat{\theta},\delta} \; \delta \quad \text{s.t.} \quad \sum_{s'} \hat{\theta}_{s'|s,a} = 1 \;\; \forall s,a \qquad \hat{\theta}_{s'|s,a} \geq 0 \;\; \forall s,a,s' \tag{6}$$

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

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

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

We propose to tackle the problem by alternating optimization where we iteratively optimize a subset of the parameters while keeping the remaining parameters fixed. We take advantage of the fact that states are organized in levels to do this. As explained earlier, states are never visited twice since at each step one more test 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 we vary only the parameters in level l while keeping the other parameters fixed, we can write the Q function of the state-action pair of any constraint before level l 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 1 describes how to compute the coefficients of the parameters at level l for the Q function

Algorithm 1 Linear dependence of the Q function at level j on the θ's at level l

```
LEVELLINEARDEPENDENCE($j, l, \pi$)
       Compute $V^\pi(s_{l+1})\ \forall s_{l+1}$
  1    $V^\pi(s_h) = R(s_h, \pi(s_h))\ \forall s_h$
  2    for $t = h-1$ down to $l+1$
  3        $V^\pi(s_t) \leftarrow R(s_t, \pi(s_t)) + \gamma \sum_{s_{t+1}} \theta_{s_{t+1}|s_t,\pi(s_t)} V^\pi(s_{t+1})\ \forall s_t$
       Initialize the coefficients for the Q function at level $l$
  4    for each $s_l, a_l$
  5        $c_{s_l,a_l}(nil) \leftarrow R(s_l, a_l)$
  6        $c_{s_l,a_l}(s_l, a_l, s_{l+1}) \leftarrow \gamma V(s_{l+1})\ \forall s_{l+1}$
  7        $c_{s_l,a_l}(s, a, s') \leftarrow 0\ \forall \langle s, a\rangle \neq \langle s_l, a_l\rangle, \forall s'$
       Compute the coefficients for the Q function at levels before $l$
  8    for $t = l-1$ down to $j$
  9        for each $s_t, a_t$
 10            $c_{s_t,a_t}(nil) \leftarrow R(s_t, \pi(s_t)) + \gamma \sum_{s_{t+1}} \theta_{s_{t+1}|s_t,\pi(s_t)} c_{s_{t+1},\pi(s_{t+1})}(nil)$
 11            $c_{s_t,a_t}(s_l, a_l, s_{l+1}) \leftarrow \gamma \sum_{s_{t+1}} \theta_{s_{t+1}|s_t,\pi(s_t)} c_{s_{t+1},\pi(s_{t+1})}(s_l, a_l, s_{l+1})\ \forall s_l, a_l, s_{l+1}$
 12    return $c$
```

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.

If we restrict the optimization problem (6) to the parameters at level l, we obtain a linear program (7) since the last constraint expresses an inequality between pairs of Q functions that are linear combinations of the coefficients at level l.

$$\max_{\hat\theta,\delta} \delta \quad \text{s.t.} \quad \sum_{s'} \hat\theta_{s_{l+1}|s_l,a_l} = 1\ \forall s, a \qquad \hat\theta_{s_{l+1}|s_l,a_l} \geq 0\ \forall s_l, a_l, s_{l+1} \tag{7}$$
$$c_{s,a^*}(nil) + \sum_{s_l,a_l,s_{l+1}} c_{s,a^*}(s_l, a_l, s_{l+1}) \hat\theta_{s_{l+1}|s_l,a_l} \geq$$
$$c_{s,a}(nil) + \sum_{s_l,a_l,s_{l+1}} c_{s,a}(s_l, a_l, s_{l+1}) \hat\theta_{s_{l+1}|s_l,a_l} + \delta \quad \forall \langle s, a^*\rangle \in \Gamma$$

To summarize, instead of directly solving the non-linear optimization problem (6), we propose an alternating optimization technique (Algorithm 2) that solves a sequence of linear programs (7) that varies 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 2 Alternating optimization to reduce the degree of inconsistency of the transition model with the expert's constraints in flat MDPs

```
ALTERNATINGOPT
1 repeat
2     Initialize $\theta$ randomly
3     repeat
4         for $l = 1$ to $h$
5             Compute coefficients for level $l$ according to Algorithm 1
6             $\delta, \{\theta_{s_{l+1}|a_l,s_l}\} \leftarrow$ solve LP (7) for level $l$
7     until convergence
8 until $\delta \geq 0$
9 return $\theta$
```

4.2 Factored Model Refinement

The approach described in the previous section assumes that we flatten the Markov decision process. This will only scale for small problems with a few test variables since the number of states grows exponentially with the number of tests. We now consider a variant for problems with a large number of tests that avoids flattening by working directly with a factored model. We assume that the transition function is 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, we assume that the parents of each variable are 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. We denote by $\theta_{X'_i|par(X_i)}$ the family of parameters defining the conditional distribution $\Pr(X'_i|par(X_i))$.

We need to deal with two issues in factored domains. First, we cannot perform dynamic programming to compute the Q-values at each state in polynomial time. We will use Monte Carlo Value Iteration [5] 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, we will partition them by families corresponding to different conditional distributions. This will allow us to alternate between a sequence of linear programs as before.

We first explain how to do approximate dynamic programming by adapting the Monte Carlo Value Iteration technique [5] (originally designed for continuous POMDPs) to factored discrete MDPs. Instead of storing an exponentially large Q-function at each stage, we store a policy graph $G = \langle N, E \rangle$. 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

Algorithm 3 Evaluate G at s

```
evalGraph($G, s$)
 1  Let $N$ be the set of nodes for $G = \langle\phi, \psi\rangle$
 2  for each $n \in N$
 3      $V(n) \leftarrow 0$
 4      repeat $k$ times
 5          $V(n) \leftarrow V(n) +$ evalTrajectory$(G, s, n)/k$
 6  $n^* \leftarrow argmax_{n \in N} V(n)$
 7  return $V(n^*)$ and $n^*$

evalTrajectory($G, s, n$)
 8  Let $G = \langle\phi, \psi\rangle$
 9  if $n$ does not have any edge
10      return $R(s, \phi(n))$
11  else
12      Sample $o \sim \Pr(o|s, \phi(n))$
13      Let $s'$ be the state reached when observing $o$ after executing $\phi(n)$ in $s$
14      return $R(s, \phi(n)) + \gamma$ evalTrajectory$(G, s', \psi(n, o))$
```

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, we will also refer to ψ 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. Algorithm 3 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, we can estimate the value of a state by calling evalGraph(G, s). While we could also use the policy graph as a controller, we will do 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 we had an explicit value function and we wanted to extract a policy. In other words, if we have a value function V, we can extract the best action a^* for any state s by computing

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

Similarly, we will extract the best action to execute at each time step when in state s based on policy graph G by computing

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

Algorithm 4 Monte Carlo Value Iteration

```
MCVI(setOfStates, horizon)
 1  Initialize $G$ with no edge and $|A|$ nodes such that $\phi$ maps each node to a different action
 2  for $t = 1$ to $horizon$
 3      for each $s \in$ setOfStates
 4          for each $a \in A$
 5              $Q(s, a) \leftarrow R(s, a)$
 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                  $[V(s'), n_{a,o}] \leftarrow$ EVALGRAPH$(G, s')$
 9                  $Q(s, a) \leftarrow Q(s, a) + \gamma Pr(o|s, a)V(s')$
10          $a^* \leftarrow argmax_a Q(s, a)$
11          Add new node $n$ to $G$ such that $\phi(n) = a^*$ and $\psi(n, o) = n_{a^*,o}$
12  return $G$
```

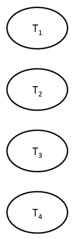

Fig. 2. Sample Policy Graph after 1 iteration of Algorithm 4

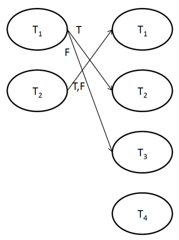

Fig. 3. Sample Policy Graph after 2 iterations of Algorithm 4

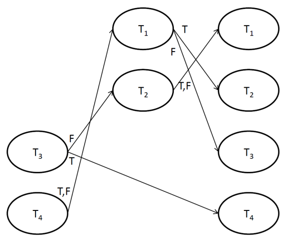

Fig. 4. Sample Policy Graph after 3 iterations of Algorithm 4

Algorithm 4 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 2, 3, and 4 present a sample trace of how the policy graph may appear after each iteration of the for loop in Algorithm 4 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.

Point-based backups are performed 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 we will see later, it is desirable to include in *setOfStates* all the states s' that are reachable from the states s for which we have constraints $\langle s, a^* \rangle$. At each iteration, a new node is added to the policy graph for each state in *setOfStates*. Although not shown in Algorithm 4, redundant nodes could be pruned from the policy graph to improve efficiency.

Algorithm 5 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  $c(nil) \leftarrow R(s,a)$ and $c(o,x) \leftarrow 0\ \forall o \in dom(X_i'), v \in dom(par(X_i))$
 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      $c(nil) \leftarrow R(s, \phi(n))$
 9      $c(o,x) \leftarrow 0\ \forall o \in dom(V_i'), v \in dom(par(V_i'))$
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      $c(nil) \leftarrow R(s,\phi(n)) + c(nil)$
22  return $c$
```

Similar to flat MDPs, we would like to optimize the parameters of the conditional distributions to satisfy the expert's constraints. We can approximate the Q-values on which we have constraints by the EVALGRAPH procedure.

$$Q^G(s,a) = R(s,a) + \gamma \sum_{s'} \Pr(s'|s,a)\text{EVALGRAPH(G,S')}$$

Since the Q-function has a non-linear dependence on the transition parameters, we partition the parameters 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, we can write the Q function as a linear combination of the parameters of a given family

$$Q(s,a) = c(nil) + \sum_{o,x} c(o,x) \Pr(X_i' = o | par(X_i) = x) \tag{8}$$

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 5 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 (7) for flat MDPs, we can define a linear program to optimize the transition parameters of a single family subject to linear constraints on Q-values as defined in Equation 8. We can also alternate between the optimization of different families similar to Algorithm 2, but for factored MDPs.

5 Evaluation and Experiments

5.1 Evaluation Criteria

Formally, for M, the true MDP that we aim to learn, the optimal policy π^* determines the choice of best next test as the one with the highest value function. If the correct choice for the next test is known (such as demonstrated by an expert), we can use this information to include a constraint on the model. We denote by $\mathbf{\Gamma}^+$ the set of observed constraints and by $\mathbf{\Gamma}^*$ the set of all possible constraints that hold for M. Having only observed $\mathbf{\Gamma}^+$, our technique will consider any $M^+ \in \mathbf{M}^+$ as a possible true model, where $\mathbf{M}^+$ is the set of all models that obey $\mathbf{\Gamma}^+$. We denote by $\mathbf{M}$ the set of all models that are *constraint equivalent* to M (i.e., obey $\mathbf{\Gamma}^*$), by $\tilde{M}$ the initial model that we start with, and by $\hat{M}_{\mathbf{\Gamma}^+}$ the particular model obtained by iterative model refinement based on the constraints $\mathbf{\Gamma}^+$.

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

Given a consistent set of constraints $\mathbf{\Gamma}$ and sufficient time (for random restarts), our 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), our approach minimizes the violation of the constraints as much as possible through alternating optimization combined with random restarts. We report the best solution found after exhausting the time quota to perform refinement.

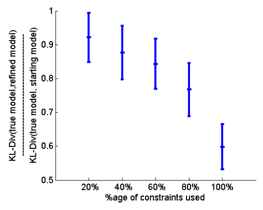

Fig. 5. Ratio of KL-divergence – Synthetic Problem

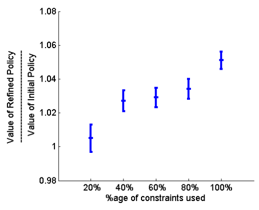

Fig. 6. Ratio of Policy Value – Synthetic Problem

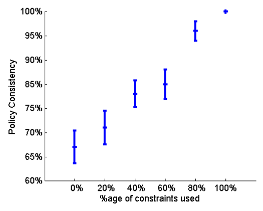

Fig. 7. Policy Consistency of Refined Model – Synthetic Problem

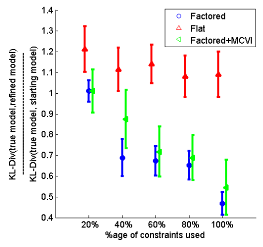

Fig. 8. Ratio of KL-divergence – Diagnostic Problem

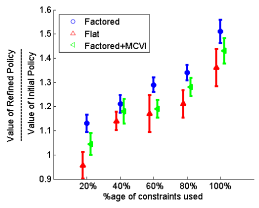

Fig. 9. Ratio of Policy Value – Diagnostic Problem

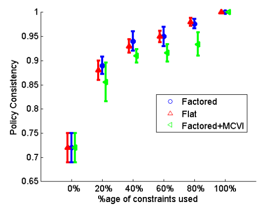

Fig. 10. Policy Consistency of Refined Model – Diagnostic Problem

5.2 Experimental Results on Synthetic Problems

We start by presenting our results on a 4-test recommender system. We want to discover the transition model of some model $M \in \mathbf{M}$. We select M by randomly sampling its transition and reward functions. Given this model M, we sample a set of constraints $\mathbf{\Gamma}^{+}$ and use our technique to find $\hat{M}_{\mathbf{\Gamma}^{+}}$. To evaluate $\hat{M}_{\mathbf{\Gamma}^{+}}$, we first compute the constraints $\mathbf{\Gamma}^{*}$ for M and estimate the set of constraint-equivalent models $\mathbf{M}$ by sampling 100 models from $\mathbf{M}$. We then compare these constraint equivalent models with $\hat{M}_{\mathbf{\Gamma}^{+}}$.

We compute the KL-divergence between each constraint-equivalent model and the refined model KL-DIV$(M_i, \hat{M}_{\mathbf{\Gamma}^{+}})$, and take its ratio with the KL-divergence between the constraint equivalent model and the initial model KL-DIV$(M_i, \tilde{M})$ as shown in Figure 5. 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 $\tilde{M}$. We can also see that the mean KL-divergence decreases as the number of constraints in $\mathbf{\Gamma}^{+}$ increases since the feasible region becomes smaller. Figures 6 and 7 show similar trends for test consistency and simulated value of the policy. We observed similar trends for KL-divergence, test consistency and simulated value of policy when increasing the number of variables.

5.3 Experimental Results on Diagnostic Problems

We also evaluate our technique on diagnostic MDPs. To construct such MDPs, we choose the number of tests and causes. The total number of actions in the MDP is 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 as executing a test only affects a part of the state space. While 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.

We present the results of model refinement 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 8, 9, and 10. These results are shown for a 4-cause and 4-test network. We see 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. We also see 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, we 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, we are using MCVI as a method to solve factored MDPs and demonstrate our technique for refinement on a large problem. We leave the question of determining an optimal $setOfStates$ for MCVI as future work, though we note that this question has been extensively studied in point-based value iteration algorithms for POMDPs [14].

5.4 Experimental Results on Large Scale Diagnostic Problems

We evaluate our technique on a real-world diagnostic network collected and reported by Agosta et al. [3], 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.

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. We pruned our diagnostic network to remove the sub-networks with no constraints to get 54 tests and 30 causes, divided in 7 sub-networks.We apply our model refinement technique to learn the parameters for each sub-network separately. The largest sub-network has 15 tests and 10 causes resulting in 25 actions and more than 14 million states. We use MCVI for these larger networks as it would not be possible to solve them exactly otherwise.

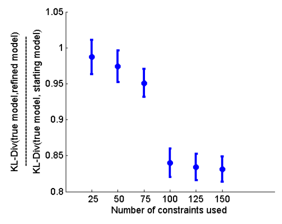

Fig. 11. Ratio of KL-divergence – Large Scale Diagnostic Problem

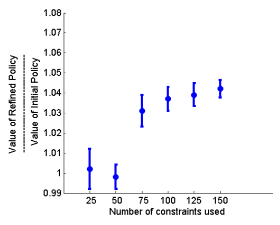

Fig. 12. Ratio of Policy Value – Large Scale Diagnostic Problem

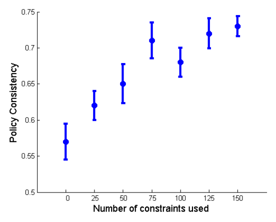

Fig. 13. Policy Consistency of Refined Model – Large Scale Diagnostic Problem

We use the 32 constraints extracted from the session logs to represent a feasible region from which we sample 100 true models. We sample 1000 states in addition to the states reachable by the constraints to form the $setOfStates$ used by MCVI. The approximation in MCVI often results in situations where no feasible model is available during refinement. In such a case, we stop the experiments after an allocated amount of time and report the model that violates the constraints the least among those computed so far. 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 11, 12, and 13 show the results for KL-divergence, simulated value of policy and policy consistency respectively for the real world diagnostic network provided by our industrial partner. Since the total number of constraints is exponential, we randomly sampled a subset of constraints and show the results 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. We can see 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.

6 Conclusion and Future Work

In summary, we presented an approach to refine the transition function of an MDP based on feedback from an expert. While several approaches address the problem of learning the reward function based on expert knowledge, this paper makes a novel contribution by tackling the problem of refining transition functions. This is particularly useful in scenarios where the amount of data (state-action-state triples) is limited. Our work makes three important contributions. First, we demonstrate how to use feedback from an expert to define constraints on the parameters of the transition function. This feedback may be implicit when obtained from logs of diagnostic sessions performed by a domain expert. Second, we design an approach to handle non-convex constraints that arise when expert feedback on optimal actions for different states is available. Third, our approach is easily applicable for flat and factored MDPs, and we demonstrate that it can be used in conjunction with approximate Monte Carlo techniques that are necessary to solve large real-world MDPs. We present results of refined models for synthetic recommender systems and a real-world diagnostic scenario from the manufacturing domain. We show that our technique not only helps in getting closer to the true transition function, but also improves policy consistency and the value of the policy.

In the future, it would be interesting to generalize this work to Partially Observable MDPs and see if the transition and observation functions can be refined simultaneously. Another possibility is to estimate transition functions from both Q-value constraints implied by user feedback and observed state transitions (i.e., state-action-state triples) by combining this work with model-based reinforcement learning approaches.

Acknowledgements

This work was supported by a grant from Intel Corporation.

References

1. Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Twenty-First International Conference on Machine Learning (ICML)*, 2004.
2. Pieter Abbeel and Andrew Y. Ng. Exploration and apprenticeship learning in reinforcement learning. In *Twenty Second International Conference on Machine Learning (ICML)*, 2005.
3. 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)*, 2010.
4. Eitan Altman. *Constrained Markov Decision Processes*. CRC Press, 1999.
5. 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.
6. Cassio P. de Campos and Qiang Ji. Improving Bayesian network parameter learning using constraints. In *International Conference in Pattern Recognition (ICPR)*, 2008.
7. Peter Geibel. Reinforcement learning for MDPs with constraints. In *European Conference on Machine Learning (ECML)*, 2006.
8. Omar Zia Khan, Pascal Poupart, and John Mark Agosta. Automated refinement of Bayes networks' parameters based on test ordering constraints. In *Neural Information Processing Systems (NIPS)*, 2011.
9. Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *Seventeenth International Conference on Machine Learning (ICML)*, 2000.
10. 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.
11. Bob Price and Craig Boutilier. Accelerating reinforcement learning through implicit imitation. *Journal of Artificial Intelligence Research (JAIR)*, 19:569–629, 2003.
12. Nathan Ratliff, J. Andrew (Drew) Bagnell, and Martin Zinkevich. Maximum margin planning. In *Twenty Third International Conference on Machine Learning (ICML)*, 2006.
13. Kevin Regan and Craig Boutilier. Robust policy computation in reward-uncertain MDPs using nondominated policies. In *Twenty-Fourth Conference on Artificial Intelligence (AAAI)*, 2010.
14. Guy Shani, Joelle Pineau, and Robert Kaplow. A survey of point-based POMDP solvers. *Autonomous Agents and Multi-Agent Systems*, 27:1–51, 2013.