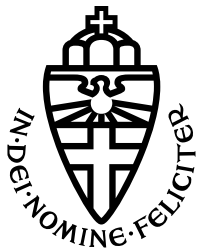
Radboud University Nijmegen



Faculty of Science

History-based Rewards for POMDPs

-

Thesis MSc Computing Science

Author:
Serena Rietbergen

Supervisor:
dr. Nils Jansen

Second reader:
- -

-

Contents

Abstract

Chapter 1

Introduction

Motivating Example

Problem Formulation

Given a POMDP with a history-based reward function, obtain a policy that maximizes the expected reward.

Contribution

Structure

Chapter 2

Preliminaries

Set Theory

Let S be any countable set, then $|S|$ denotes the cardinality. We let S^* and S^ω denote the set of finite and infinite sequences over S, respectively. For a sequence $\pi \in S^*$ we can denote the length by $|\pi|$.

Let $\epsilon \in S^*$ be the sequence of length zero, so $|\epsilon| = 0$.

TO WRITE: sequence with set length k will be denoted by S^k, of which an element can be rewritten to $s_1 s_2 \dots s_k$

Probability Theory

For any countable set S we can define a *discrete probability distribution* as $\psi : S \to [0, 1]$ where $\sum_{s \in S} \psi(s) = 1$. The set of all possible probability distributions over S is denoted as $\Pi(S)$. We denote the support of a *probability distribution* as $supp(\psi) = \{s \in S \mid \psi(s) > 0\}$.

TO WRITE: random variable, expected value

Chapter 3

Background

3.1 Markov decision processes

TO WRITE: introduction to MDP

Definition 3.1 (MDP)**.** A Markov decision process is a tuple $M = (S, s_I, A, T)$ where

- S, the finite set of states;
- $s_I \in S$, the initial state;
- A, the finite set of actions;
- $T : S \times A \to \Pi(S)$, the probabilistic transition function.

Note that given $s \in S, a \in A$, we assign a probability distribution over S through $T(s, a)$. To obtain the probability of ending up in a certain state s' when starting in state s and performing action a, we simply calculate $T(s, a, s')$ which we obtain through $T(s, a)(s')$.

The *available actions* for a state s are given by $A(s) = \{a \in A \mid \exists s' \in S : T(s, a, s') > 0\}$. We can give the *possible successors* of state s in a similar matter through $Succ(s) = \{s \in S \mid \exists a \in A : T(s, a, s') > 0\}$.

A finite *trajectory* or *run* π of an MDP is realization of the stochastic process performed by the MDP denoted by the finite sequence $s_1 a_1 s_2 a_2 \dots s_{n-1} a_{n-1} s_n \in (S \times A)^* \times S$. To obtain the last state of a trajectory we can use the following

$$last(\pi) = last(s_1 a_1 s_2 a_2 \dots s_{n-1} a_{n-1} s_n) = s_n$$

TO WRITE: Set notation for finite trajectories

Reward function

We can extend MDPs with a *reward function* R which assign a reward - usually in $\mathbb{R}$ for taking a certain action a in a state s.

TO WRITE: Intuitive explanation for reward function - including cost function, including a real-world example

Let us look at *Markovian reward functions*, which can determine a reward based on the current state, action and obtained state, independent of its history. The most conventional notation is $R : S \times A \to \mathbb{R}$, where we consider the current state and the taken action. Another possible definition is $R : S \times A \times S \to \mathbb{R}$, where in $R(s, a, s')$ we consider the specific transition from s to s' by using action a, or $R : S \to \mathbb{R}$ where in $R(s)$ we only consider the states visited.

TO WRITE: Real life example of reward function with history

A reward function which is dependent of its history is called a *Non-Markovian reward function*. There are a number of different reward functions possible

- $R : S^* \to \mathbb{R}$ - which only looks at the finite states visited, or;
- $R : (S \times A)^* \to \mathbb{R}$ - which looks at the finite (sub)trajectory without the last state, or;
- $R : (S \times A)^* \times S \to \mathbb{R}$ - which looks at the finite (sub)trajectory.

The reward function we will be using is the Non-Markovian reward function which looks at trajectories of specific length k, namely $R_k : (S \times A)^k \to \mathbb{R}$.

TO WRITE: Increasing k creates increased reward

Policy

As stated above, we use reward functions over a MDP to usually argue over an optimized expected reward. After retrieving such an optimum, the question remains on how to actually obtain this value. We wish to know what strategy to apply path to take to obtain this value. For this we use strategies, or often called policies.

Definition 3.2. A policy for a MDP M is a function $\sigma : (S \times A)^* \times S \to \Pi(A)$, which maps a trajectory π to a probability distribution over all actions.

We call a policy *memoryless* if the function only considers $last(\pi)$. The notation $\sigma_k : (S \times A)^{k-1} \times S \to \Pi(S)$, gives a probability distribution when given the trajectory of length k - the k visited states and $k - 1$ actions. $\sigma_k(\pi, a) = \sigma_k(\pi)(a)$ gives the probability of using action a after the trajectory π.

TO WRITE: induced markov chain for removing non-determinism

3.2 Solve for optimal policy

Explain here or refer to paper(s)?

Value Iteration

Policy Iteration

Linear Programming

3.3 Partial observability

TO WRITE: Introduce pomdp

Definition 3.3 (POMDP)**.** A partially observable Markov decision process (POMDP) is a tuple $\mathcal{M} = (M, \Omega, O)$ where

- $M = (S, s_I, A, T)$, the hidden MDP;
- Ω, the finite set of observations;
- $O : S \to \Omega$, the observation function.

Let $O^{-1} : \Omega \to 2^S$ be the inverse function of the observation function, like so $O^{-1}(o) = \{s \in | \ O(s) = o\}$ in which we simply obtain all states in S that have observation o. Without loss of generality we assume that states with the same observations have the same set of available actions, thus $O(s_1) = O(s_2) \Rightarrow A(s_1) = A(s_2)$.

Since the actual states in a trajectory of the hidden MDP are not visible to the observes, we argue about an *observed trajectory* of the POMDP $\mathcal{M}$. This is not consist of a sequence of states and actions, but instead a sequence of observations are actions, thus an element of $(\Omega \times A)^* \times \Omega$. The set of all possible finite observed trajectories of will be denoted as $ObsSeq^{\mathcal{M}}$.

We can argue about the observed trajectory through the observation function, which will be extended over trajectories, like so

$$O(\pi) = O(s_1 a_1 s_2 a_2 \dots s_{n-1} a_{n-1} s_n) = O(s_1) a_1 O(s_2) a_2 \dots O(s_{n-1}) a_{n-1} O(s_n)$$

Policy

Definition 3.4. An observation-based strategy of a POMDP $\mathcal{M}$ is a function $\sigma : ObsSeq^{\mathcal{M}} \to \Pi(A)$ such that $supp(\sigma(O(\pi))) \subseteq A(last(\pi)) \ \forall \pi \in (S \times A)^* \times S$.

Belief MDP

3.4 Moore machine

Based on the definition as presented in [1].

Definition 3.5. A Mealy machine is a tuple $(Q, q_0, \Sigma, O, \delta, \sigma)$ where

- Q, the finite set of states;
- $q_0 \in Q$, the initial state;
- Σ, the finite set of input characters - the input alphabet;

- O, the finite set of output characters - the output alphabet;
- $\delta : Q \times \Sigma \to Q$, the input transition function, and;
- $\sigma : Q \times O$, the output transition function.

Example

TO WRITE: example moore machine – traditional sense

Chapter 4

Reward Controllers

4.1 Definition

TO WRITE: introduction

The idea is to transform the history-based reward function into something more tangible. We transform it is so that we can obtain the reward per step instead of only at the end of a sequence.

Based on the history-based reward function $R : \Omega^* \to \mathbb{R}$ of a POMDP $\mathcal{M}$, we build a reward controller that mimics its behavior.

Definition 4.1. A reward controller $\mathcal{F}$ is a reward machine $(N, n_I, \Omega, \mathbb{R}, \delta, \lambda)$, where

- N, the finite set of memory nodes;
- $n_I \in N$, the initial memory node;
- $\Omega \times A$, the input alphabet;
- $\mathbb{R}$, the output alphabet;
- $\delta : N \times \Omega \to N$, the memory update;
- $\sigma : N \times \Omega \to \mathbb{R}$, the reward output.

4.2 Creating a reward controller from a list of sequences

dit kan wel, maar moet aangepast worden voor gegeven reeks inputs

Starting from a fully known R_k and POMDP $\mathcal{M}$, we know that R is defined over a number of sequences of elements in Ω^*. Since the reward of a sequence is only obtained after running through the entire sequence, we encode the reward of that specific sequence into the last transition. The idea is to make sure that every unique sequence has an unique path to the final state, so that their own reward is obtained.

rewrite entire section lmao

Given all the sequences over which the Non-Markovian reward function is defined, let us create a reward controller through the following procedure.

Algorithm 1 Procedure for turning a list of sequences into a reward controller

1: **procedure** CreateRewardController(sequences, R)
Require: sequences
Require: $R : \Omega^* \to \mathbb{R}$
2: $n_I \leftarrow$ `new Node()` ▷ initial node
3: $n_F \leftarrow$ `new Node()` ▷ dump node
4: $N \leftarrow \{n_I, n_F\}$
5: $\sigma(n_I), \sigma(n_F) \leftarrow 0$
6: **for all** $\pi = o_1 o_2 \dots o_k$ in sequences **do**
7: $n \leftarrow n_I$
8: **for** $i \leftarrow 1, \dots, k$ **do**
9: **if** $\delta(n, o_i)$ is undefined **then**
10: $n' \leftarrow$ `new Node()` ▷ create new memory node
11: $\sigma(n) \leftarrow 0$
12: $N \leftarrow N \cup \{n'\}$
13: $\delta(n, o_i) \leftarrow n'$
14: $n \leftarrow \delta(n, o_i)$ ▷ update memory node
15: $\sigma(n) \leftarrow \sigma(n) + R(\pi)$ ▷ set reward
16: **for all** $n \in N$ **do** ▷ makes δ and σ deterministic
17: **for all** $o \in \Omega$ **do**
18: **if** $\delta(n, o)$ is undefined **then** ▷ useless transition
19: $\delta(n, o) \leftarrow n_F$
20: **return** $(N, n_I, \Omega, \mathbb{R}, \delta, \sigma)$

TO WRITE: initialisation and walking through file

TO WRITE: making deterministic

We observe that the number of memory nodes $|N|$ of the newly created reward controller $\mathcal{F}$ is bounded by Ω^k, where $k = \max_{seq \in sequences} |seq|$.

Example

Let R be defined over the following sequences

- □ □ □
- □ □ ■
- □ ■ ■

TO WRITE: Notes for the model: missing actions/states

Following the procedure 1 we can create the reward controller as seen in Figure **??**. Note that everywhere where the red lines are depicted, those paths are obsolete. And following the procedure those end-markings should be transitions towards the memory node they originated from. The green highlighted are the rewards that are set in Line 15. The elaborated version shown in the figure first checks the observation and then check the action to then direct it to a new memory node.

A more compact version can be seen in Figure **??**, where the observation and the action are condensed into one transition. Note that in this figure, the self-loops are also not shown.

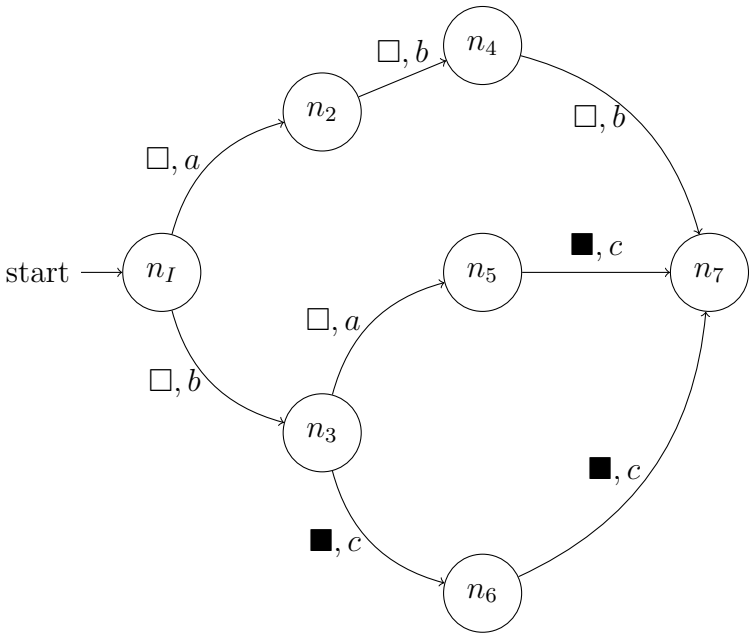


Figure 4.1: Compact reward controller

Corollary 4.1.1. *For every path $\pi \in \Omega^*$ that is defined for R_k, we have that $R_k(\pi) =$ `FinalReward`(n_I, π).*

4.3 Creating a Reward Controller from a sequence of regular expressions

find ref for product construction

Given a number of expressions over which R is defined as $e_1, e_2, \ldots, e_n$, which are regular expressions over observations. For all $i = \{1, \ldots, n\}$, from the regular expression e_i create a DFA $M_i = (Q_i, q_{0,i}, \Omega, \delta_i, F_i)$. From all these seperate DFAs, we now create the product DFA, which encapsulates the entire problem as follows. We obtain the product DFA $N = (Q, q_0, \Sigma, \delta, F)$ where

- $Q = Q_1 \times Q_2 \times \cdots \times Q_n$
- $Q = \langle q_{0,1}, q_{0,2}, \ldots, q_{0,n} \rangle$
- Ω, the same input alphabet
- $\delta(\langle q_1, q_2, \ldots, q_n \rangle, a) = \langle \delta_1(q_1, a), \delta_2(q_2, a), \ldots, \delta_n(q_n, a) \rangle$
- $F = \{\langle q_1, q_2, \ldots, q_n \rangle \mid \exists i \in \{1, 2, \ldots, n\} : q_i \in F_i\}$

TO WRITE: R_A keeps track of all the rewards associated with their respective regular expression

$$R_A(q) = \begin{cases} R(e_i) & \text{if } q \in F_i \\ 0 & \text{otherwise} \end{cases}$$

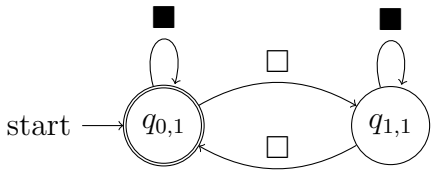
Definition 4.2. So having this DFA $N = (Q, q_0, \Omega, \delta, F)$ and associated reward function R_A, we define the induced reward controller $mathcalF = (N, n_I, \Omega, \mathbb{R}, \delta_{\mathcal{F}}, \sigma)$ as follows

- $N = Q$
- $n_I = q_0$
- $\delta_{\mathcal{F}} = \delta$
- $\sigma : Q \to \mathbb{R}$ where $\sigma(\langle q_1, q_2, \ldots, q_n \rangle) = \sum_{i=1}^{n} R_A(q_i)$

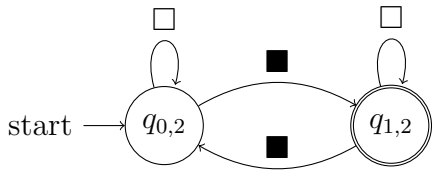
4.3.1 Example

Let's say an even number off □ gives a reward of 10 and an uneven number of ■ gives a reward of 15. In other words $R((■^*□■^*□■^*)^*) = 10$ and $R(□^*■□^*(■□^*■□^*)^*) = 15$

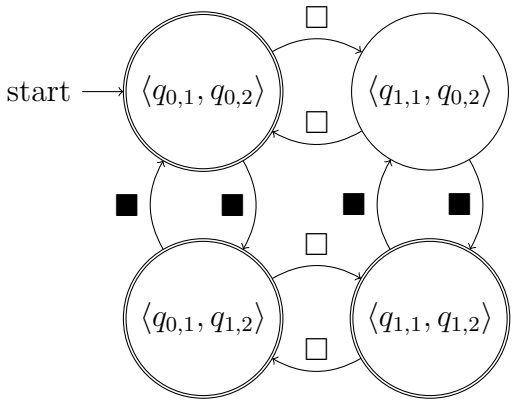
are the regular expressions really neccesary?



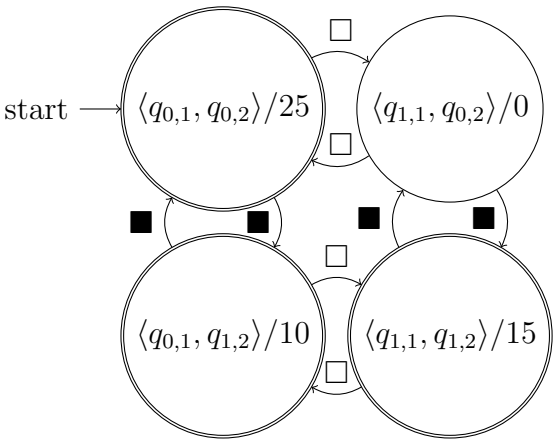
(a) DFA for regular expression even number of □



(b) DFA for regular expression for odd number of ■



(c) Product DFA for both regular expressions



(d) Reward Controller for R

Note that
$R_A(q_{0,1}) = 10$
$R_A(q_{1,1}) = 0$
$R_A(q_{0,2}) = 0$
$R_A(q_{1,2}) = 15$

Chapter 5

Obtaining policy

TO WRITE: introduction

Definition 5.1. The induced POMDP for reward controller $\mathcal{F} = (N, n_I, \Omega, \mathcal{R}, \delta, \lambda)$ on a POMDP $\mathcal{M} = (M, \Omega, Obs)$ where $M = (S, s_I, A, T_M)$ is a tuple $\mathcal{M}' = (M', \Omega', Obs')$ where

- $M' = (S', s'_I, A', T_{M'}, \mathcal{R})$, the hidden MDP defined as follows
  - $S' = S \times N \cup \{s_F\}$
  - $s'_I = \langle s_I, \delta(n_I, O(s_I))\rangle$
  - $A' = A \cup \{\texttt{end}\}$
  - $T_{M'} : S' \times A \to \Pi(S)$ where

$$T_{M'}(\langle s, n\rangle, a, \langle s', n'\rangle) = \begin{cases} T_M(s, a, s') & \text{if } \delta(n, O(s') = n') \\ 0 & \text{otherwise} \end{cases}$$

$$T_{M'}(s, \texttt{end}, s_F) = 1 \text{ for all} s \in S'$$

  - $R : S' \times A' \times S' \to \mathcal{R}$ where

$$R(s', \texttt{end}, s_F) = \begin{cases} \sigma(n) & \text{if } s' = \langle s, n\rangle \\ 0 & \text{if } s' = s_F \end{cases}$$

$$R(s, a, s') = 0 \text{ for all } s' \in S' \setminus \{s_F\}$$

- $\Omega' = \Omega \cup \{o_F\}$, the observation state
- $Obs : S' \to \Omega'$ where

$$Obs'(s) = \begin{cases} Obs(s') & \text{if } s = \langle s', n\rangle \\ o_F & \text{if } s = s_F \end{cases}$$

Bibliography

[1] Edward F. Moore. Gedanken-experiments on sequential machines. In Claude Shannon and John McCarthy, editors, *Automata Studies*, pages 129–153. Princeton University Press, Princeton, NJ, 1956.