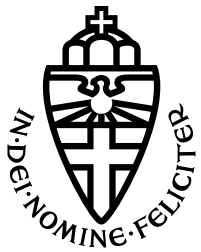
Radboud University Nijmegen



Faculty of Science

Modelling Non-Markovian Properties

-

Thesis MSc Computing Science

Author:
Serena Rietbergen

Supervisor:
dr. Nils Jansen

Second reader:
- -

-

Contents

Abstract

hoi hier moet iets

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 $|\pi|$.

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

Markov decision processes

TO WRITE: explain in words what an mdp is

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

- S, the finite set of states;
- s_I, the initial state;
- A, the finite set of actions;
- $T_M : S \times A \to \Pi(S)$, 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 $Succ(s) = \{s \in S | \exists a \in A : T(s, a, s') > 0\}$.

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

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 $R : A \to \mathbb{R}$ where in $R(a)$ we only consider the action taken.

TO WRITE: Real life example of markovian reward function

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

- $R : A^* \to \mathbb{R}$ - which only needs all the finite actions used, or;
- $R : S^* \to \mathbb{R}$ - 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.

All of these reward functions can be extended to the infinite sequences.

infinite needed?

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, which includes 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??

Solve optimal policy

Explain here or refer to paper(s)?

Value Iteration

Policy Iteration

Linear Programming

Partial observability

TO WRITE: Introduce pomdp

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

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

Without loss of generality we assume that states with the same observations have the same set of available actions, thus $O(s_1) = O(s_2) \Rightarrow A(s_1) = A(s_2)$.

TO WRITE: O^{-1}

Since the actual states in a trajectory of the hidden MDP are not visible to the observes, we argue about an *obversed 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 $(Z \times A)^* \times Z$. 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 $supp(\sigma(O(\pi))) \subseteq A(last(\pi))\ \forall \pi \in (S \times A)^* \times S$.

Mealy machine

Chapter 4

Reward Controllers

4.1 Definition

Idea: transform the history-based reward function into something more tangible. Problem: we can only work with the observations, since we are working with a pomdp.

Based on the history-based reward function $R_k : (S \times A)^k \to \mathbb{R}$ of a POMDP $\mathcal{M}$, we build a reward controller that mimics its behavior.

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

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

When starting in memory node n, where we observe observation o and then perform Aion a we will end up in the new memory node $\delta(n, o, a)$.

When working with the sequence $s_0 a_0 s_1 a_1 \dots s_{k-1} a_{k-1} s_k \in (S \times A)^k \times S$ we also obtain two sequences from the connected R-FSC $\mathcal{F}$. First we obtain the memory sequence $n_0 n_1 \dots n_{k-1}$, where $n_0 = n_I$ and n_{i+1} is obtained with probability $\delta(n_i, O(n_i), a_i, n_{i+1})$. Second, we obtain the reward sequence $r_0 r_1 \dots r_{k-1}$. In this reward sequence we claim that $r_i = R(n_i, O(s_i), a_i, n_{i+1})$.

Since $\mathcal{F}$ simulates R_k of $\mathcal{M}$, we want the reward of R_k and of $\mathcal{F}$ to be the same when using the same trajectory. Thus $R_k(s_1 a_1 \dots s_k a_k) = \mathcal{R}(n_{k-1}, O(s_k), a_k, n_k) = r_k$.

4.2 Creating a reward controller

Starting from a fully known R_k and POMDP $\mathcal{M}$, we know that R_k is defined over a number of sequences of elements in $Z \times A$ with length k.

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

procedure CREATEREWARDCONTROLLER(sequences, R_k)
Require: sequences $\in 2^{(Z\times A)^k}$
Require: $R_k \in 2^{(Z\times A)^k \to \mathbb{R}}$
    $n_I =$ `new Node()` ▷ initial node
    $n_F =$ `new Node()` ▷ "final" node
    $N = \{n_I, n_F\}$
    for all $\pi = \pi_1\pi_2 \dots \pi_k$ in sequences **do**
        $n = n_I$
        for $i = 1$ to $k - 1$ **do**
            if $\delta(n, \pi_i)$ is undefined **then**
                $n' =$ `new Node()` ▷ create new memory node
                $N = N \cup \{n'\}$
                $\delta(n, \pi_i) = n'$
            $n = \delta(n, \pi_i)$ ▷ update memory node
        $\delta(n, \pi_k) = n_F$ ▷ set final transitions
        $\lambda(n, \pi_k) = R_k(\pi)$ ▷ set reward
    for all $n \in N$ **do** ▷ makes δ and λ deterministic
        for all $(o, a) \in (Z \times A)$ **do**
            if $\delta(n, (o, a))$ is undefined **then** ▷ self-loop
                $\delta(n, (o, a)) = n$
            if $\lambda(n, (o, a))$ is undefined **then ??**
                $\delta(n, (o, a)) = 0$
    return $(N, n_I, (Z \times A), \mathbb{R}, \delta, \lambda)$

- Every path should have an unique transition to n_F, for that is the transition in which the reward is encoded. - we assume all paths are unique, so for every sequence we will create at least one new node. - if sequences share the first few entries, only from the moment they differ a new node will appear.

TO WRITE: In words what happens

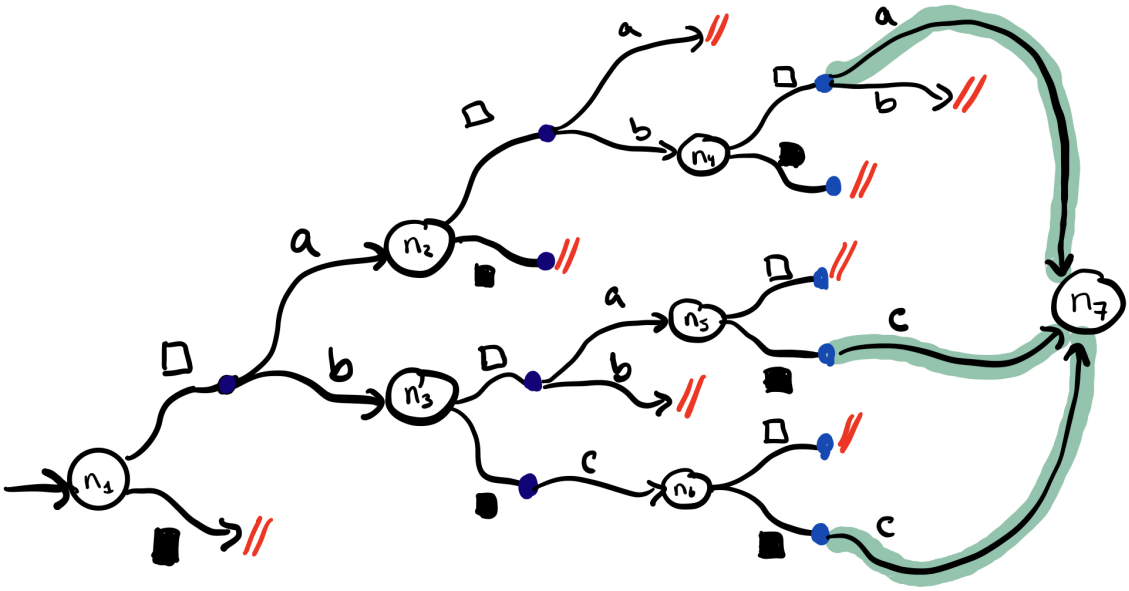
Example

Let R_k be defined over the following sequences

- $\square\ a\ \square\ b\ \square\ c$
- $\square\ b\ \square\ a\ \blacksquare\ c$
- $\square\ b\ \blacksquare\ c\ \blacksquare\ c$

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

For this model, everywhere where the red lines are, is an transition towards the node it originated from. - the highlighted green line contains the reward.



Or more simplified - only the transitions to n_7 are encoded with the reward, the rest remain zero as stated in Line **??**

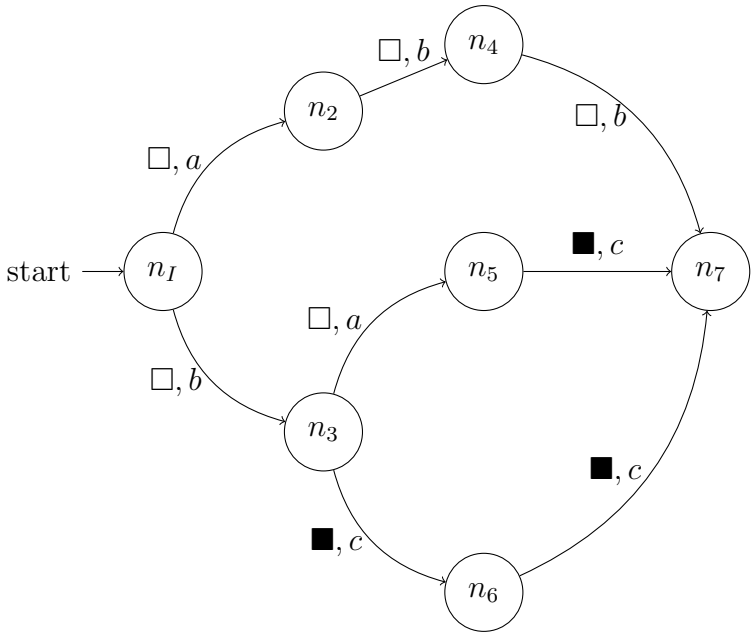


Figure 4.1: Simplified reward controller

Chapter 5

Obtaining policy

TO WRITE: introduction for why/what reward mdp is needed

create product of reward controller and pomdp known: - read sequences of length k - only know observations

do: - instead of history reward, we encode the reward in the new model per step, so that $R : V \times A \to \mathbb{R}$ instead of $R_k : (Z \times A)^k \to \mathbb{R}$. this allows us to use traditional methods for solving it. - create a limited belief state mdp, since we know s_0

Definition 5.1. The induced Reward Markov Decision Process for reward controller $\mathbb{F} = (N, n_I, A, \mathbb{R}, \delta, \lambda)$ on a POMDP $\mathcal{M} = (M, Z, O)$ where $M = (S, s_I, A, T_M)$ extended with R_k is a tuple $\mathcal{A} = (V, v_I, A, T_\mathcal{A}, \mathcal{R})$ where

- $V = Z \times N$, the finite set of states;
- $v_I = \langle O(s_I), n_I \rangle$, the initial state;
- A, the available actions;
- $T_\mathcal{A} : V \times A \to \Pi(V)$ where

$$T_\mathcal{A}(\langle o, n\rangle, a)(\langle o', n'\rangle) = \begin{cases} \frac{\sum_{s \in O^{-1}(o)} \sum_{s' \in O^{-1}(o')} T_\mathcal{M}(s,a,s')}{\sum_{s \in O^{-1}(o)} \sum_{s' \in S} T_\mathcal{M}(s,a,s')} & \text{if } \delta(n, o, a) = n' \\ 0 & \text{otherwise} \end{cases}$$

- $\mathcal{R} : V \times A \to \mathbb{R}$ defined as $\mathcal{R}(\langle o, n\rangle, a) = \lambda(n, (o, a))$

The different transitions to the final state are irrelevant, since the reward is encoded in the action, not including the obtained state

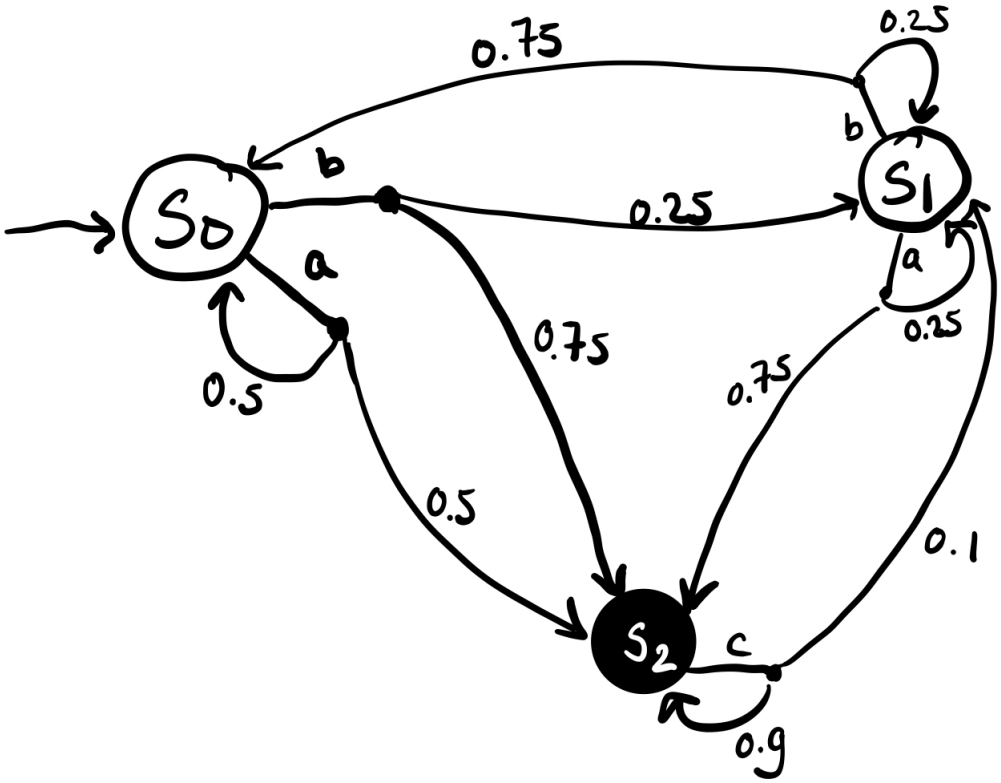
After obtaining this reward MDP, which now has a Markovian reward function since it's only dependent of the state it's in and the action following, we can now find and optimal policy through known methods as discussed in .

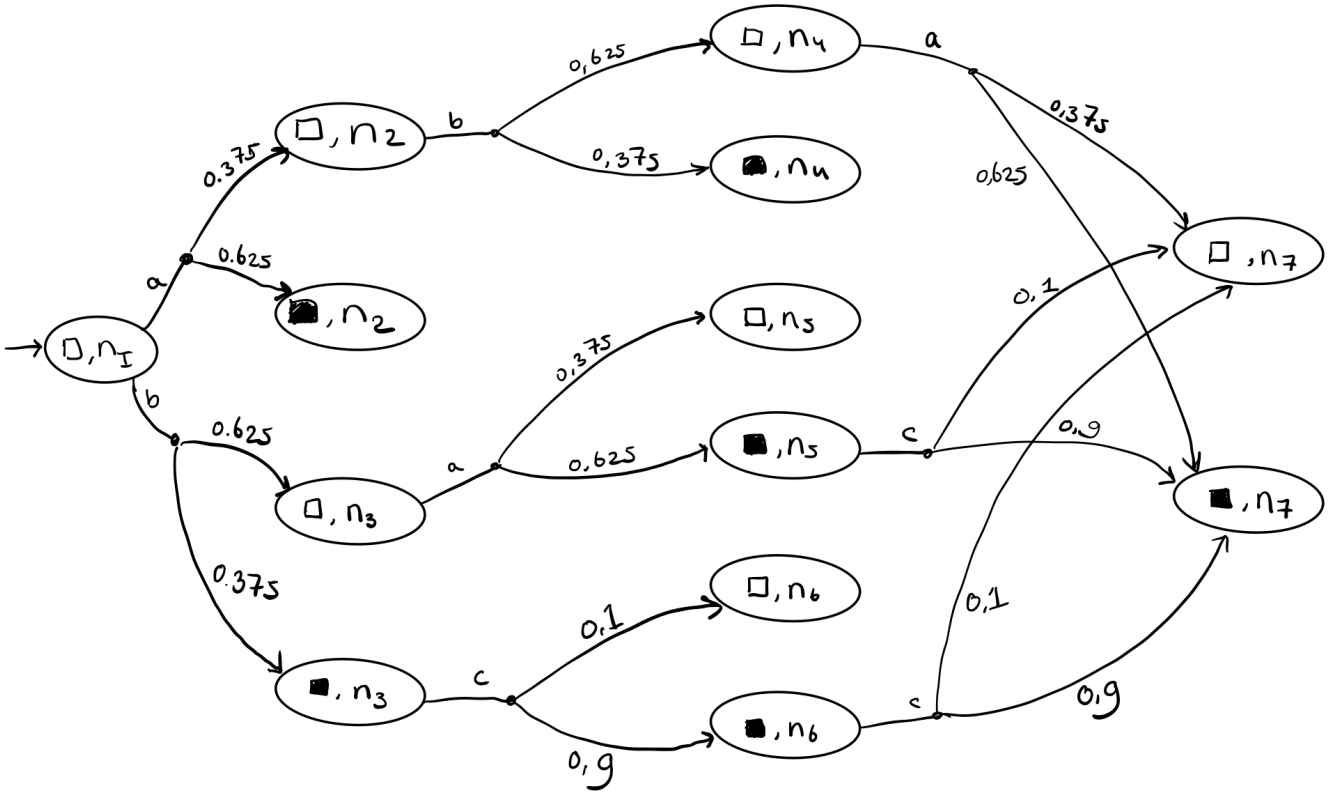
TO WRITE: After obtaining this, we can obtain an optimal policy

Corollary 5.1.1. *Given a POMDP $\mathcal{M}$ and the reward controller $\mathcal{F}$ obtained through the history-based reward function R_k of $\mathcal{M}$, we create the Reward Machine $\mathbb{A}$ with the help of Definition 5.1. The expected optimal reward of $\mathcal{A}$ is the same as of $\mathcal{M}$*

Example

Given the reward controller as found in Figure 4.2 and the POMDP with $Z = \{\square, \blacksquare\}$ as seen in Figure 5, we can create the induced reward MDP.





TO WRITE: explain the transitions

TO WRITE: explain that the missing transitions are self-loops/deadlock/not needed

Bibliography

[1] Zhe Xu, Ivan Gavran, Yousef Ahmad, Rupak Majumdar, Daniel Neider, Ufuk Topcu, and Bo Wu. Joint inference of reward machines and policies for reinforcement learning. In *Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 590–598. AAAI Press, 2020.